THE CREATION OF A FUNCTIONAL MAILING LIST SERVER WITH A
GRAPHICAL USER INTERFACE

A Thesis Presented To

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by

Brian Wilson
June 1997
ACKNOWLEDGMENTS

To my family, who has given me enough support to make it through school. I'd like to acknowledge the assistance Dr. Shawn Ostermann, Mark Allman, and the entire IRG for all their ideas and help.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Mailing List Programs</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Mailing List Functionality</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Security</td>
<td>2</td>
</tr>
<tr>
<td>2. EXISTING SOLUTIONS TO THE PROBLEM</td>
<td>4</td>
</tr>
<tr>
<td>2.1 MajorDomo</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Purdue Mailer</td>
<td>5</td>
</tr>
<tr>
<td>3. THE OU MAILER</td>
<td>6</td>
</tr>
<tr>
<td>3.1 C/UNIX Back End</td>
<td>6</td>
</tr>
<tr>
<td>3.1.1 Remote Procedure Call</td>
<td>6</td>
</tr>
<tr>
<td>3.2 GUI interface</td>
<td>8</td>
</tr>
<tr>
<td>3.3 PERL E-mail Resender</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Other Considerations</td>
<td>12</td>
</tr>
<tr>
<td>4. CONCLUSIONS</td>
<td>14</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>16</td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>A. SCREEN SHOTS</td>
<td>17</td>
</tr>
<tr>
<td>B. SOURCE CODE</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>B.1 Back End Source Code</td>
<td>19</td>
</tr>
<tr>
<td>B.1.1 Mailer.c</td>
<td>19</td>
</tr>
<tr>
<td>B.1.2 Syntax.c</td>
<td>77</td>
</tr>
<tr>
<td>B.1.3 Util.c</td>
<td>81</td>
</tr>
<tr>
<td>B.2 GUI Interface Source Code</td>
<td>99</td>
</tr>
<tr>
<td>B.2.1 AppletClient.java</td>
<td>99</td>
</tr>
<tr>
<td>B.2.2 InfoDialog.java</td>
<td>128</td>
</tr>
<tr>
<td>B.2.3 MultiLineLabel.java</td>
<td>130</td>
</tr>
<tr>
<td>B.3 GUI Server Source Code</td>
<td>133</td>
</tr>
<tr>
<td>B.3.1 KKMultiServer.java</td>
<td>133</td>
</tr>
<tr>
<td>B.3.2 KKMultiServThread.java</td>
<td>134</td>
</tr>
<tr>
<td>B.3.3 KKstate.java</td>
<td>135</td>
</tr>
<tr>
<td>B.3.4 Exec.java</td>
<td>137</td>
</tr>
<tr>
<td>B.4 PERL Resender Source Code</td>
<td>142</td>
</tr>
</tbody>
</table>

ABSTRACT ................................................. 154
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>9</td>
</tr>
<tr>
<td>3.3</td>
<td>10</td>
</tr>
<tr>
<td>3.4</td>
<td>11</td>
</tr>
<tr>
<td>3.5</td>
<td>12</td>
</tr>
<tr>
<td>A.1</td>
<td>17</td>
</tr>
<tr>
<td>A.2</td>
<td>17</td>
</tr>
<tr>
<td>A.3</td>
<td>18</td>
</tr>
<tr>
<td>A.4</td>
<td>18</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1 Mailing List Programs

On the current Internet, it is possible to send e-mail to one or many people. A computer can be set to associate a list of e-mail addresses with a single e-mail address. When a person sends e-mail to this list, the e-mail sent will get sent to everyone on this list [Pee94].

1.2 Mailing List Functionality

A mailing list should provide the following functions:

- **Public/Private Lists**: A mailing list program should make a distinction between lists that are public for anyone to join and lists that are private. It should be easy for anyone to find and join a public list.

- **Hiding Private Lists**: A private list should not only be hidden to the user, but the list's subscribers should also be kept private from the general public.

- **Exclusive Mailing Lists**: A list owner should have the right to make a list accept e-mail from only the list members so that non-members cannot send e-mail to the list.

- **Moderated Lists**: A list owner should have the ability to moderate the contents of the e-mail sent to his list; therefore every e-mail sent to that list would need to be individually approved by the list owner.
1.3 Security

Security is an important issue with any distributed application. The mailing list program needs to identify each user. If it is easy to circumvent this identification process, then a malicious user could easily pose as another user. The program should thoroughly verify the user's identity to thwart those who might try to mask their identity. If a user can fake his identity, then he could perform the following actions:

- **Delete Users From Lists**: With a fake identity, a malicious user could delete other list members.

- **Add Users To Lists**: With a fake identity, a malicious user could add an unsuspecting user to a mailing list.

- **Create Lists**: Using a fake identity, a malicious user could create mailing lists with an unsuspecting user as the maintainer.

- **Pretend To Be a List Maintainer**: Using a fake identity, a malicious user could pretend to be the maintainer of a mailing list. By doing this, the user could change the properties of the list or delete the whole list itself.

Any mailing list server in which it is easy fake a users identity will not be a secure program. It is possible to make a secure program which allows no interactions to the server to occur without complete identification and validation. There are methods, such as encrypting information with public and private keys, that would be very secure. With this method, a server would receive encrypted packets and try to decode them with the supposed users' public key. If the decryption does not work, the server can assume that the user is not who he claims to be [RSA96]. Although this method is effective, it is also complicated. Using such a system for identification would require all users to not only make public/private keys and display these keys, but it would also require the users to send all transmissions encrypted. Making our program this complicated might scare away would be users.
A password scheme, in which the user has to log onto the server using a password, does not seem as complicated. Users are used to having passwords to log onto different servers, so making them log onto a mailing list server program should not be too difficult. One problem does exist with using a password scheme; how does the user initially set his password? The user could set the password the first time he uses the program, but that still would allow some people to pretend to be others the first time they log on. The program could use a Unix password on a given system for a user. To do this the program would have to need access to the Unix password file. This file is usually closely guarded by system administrators, so getting access to the file is usually not an option. Also, most users are reluctant to use their Unix passwords for anything other than logging onto Unix machines because they want to keep their passwords a secret. Somehow the user needs to securely pre-set his password before he uses the program for the first time.

The rest of this paper discusses mailing lists, mailing list programs, and the possible interfaces for such programs. Chapter 2 focuses on the programs currently available for creating mailing lists. Chapter 3 describes how the proposed solution was implemented. Chapter 4 tells how things should work together and gives suggestions for future modifications.
2. EXISTING SOLUTIONS TO THE PROBLEM

2.1 MajorDomo

MajorDomo [Cha92] is a mailing list program written by Brent Chapman. Most of MajorDomo is written in the scripting language PERL [WS91], and it uses some C functions to do actions not possible in PERL. MajorDomo has a mail-in interface, which means that to use the program, a user has to actually send e-mail to it. This interface is extremely slow. To begin to use the program, a user has to send e-mail to it just to get directions on how to use the program. Then, while trying to join a list, a user might make a mistake. It will take quite a while just to discover the mistake because the program will have to send e-mail back to him.

To unsubscribe from a list, a user needs to send e-mail to the MajorDomo program. To do this, a user has to send the e-mail from the account that is registered to the list. It is possible to join a mailing list through MajorDomo from one account and lose that account. If the machine of the old account has set up an alias for the user, all the old e-mail will be forwarded to his new account. That means that he would still be getting the mailing list e-mail, but he would not be able to take himself off that list because he could not send e-mail from his old address to MajorDomo.

When MajorDomo gets e-mail for a certain list, it runs a PERL script to resend the e-mail. This script looks in a file of members for that list. The script then sends the e-mail to everyone in that file.
2.2 Purdue Mailer

The Purdue mailer was written by Bill Nowicki and Steve Holmes, and is was made into a distributed program by Shawn Ostermann at Purdue University. This mailing list program is written totally in C and needs to be run on a Unix platform. The Purdue mailer is run from a Unix prompt, and it works by modifying the /etc/aliases file directly.

This Unix rerouting of the e-mail is done by the Unix sendmail [Fri95] daemon. This daemon receives e-mail, and then it reads the /etc/aliases file to see if the e-mail should be handled specially. When a list made by the Purdue mailer receives e-mail, sendmail reads the aliases file and sends a copy of the e-mail to every address listed on the line for that alias. Since the program modifies the /etc/aliases file directly, it causes some problems. This limits the length of a mailing list to the length of one standard Unix line\(^1\). Since the /etc/aliases file is an important system file, the Purdue Mailer only updates the file hourly. It is possible to create a list and add members to that list, but the list still would not exist until this hourly update. Modifying the /etc/aliases file directly also allows people to see who is on private lists by logging into the mail port and running the EXPN (expand mail alias) [CDF89] command. Finally, a user must have a Unix account to use this mailer program, and this restriction severely limits the people that can use it.

\(^1\)The Purdue mailer uses a method of list member retrieval that limits the number of characters on a line. All the members of a list are kept on one line, so the number of members a list can have is limited by the maximum length of one line.
3. THE OU MAILER

In order to implement a solution to the mailing list program problem, the OU mailer combines features from the above-mentioned programs. Pieces from each program were used, in addition to a few new ideas (see figure 3.1).

3.1 C/UNIX Back End

The OU mailer program takes many variable arguments. It does all of the maintenance, creation, adding to, deleting from, listing of, and modification of the e-mail lists (see appendix B.1). It is written in C, and it uses an RPC [Sri95] network interface. RPC allows the C back end to run on a computer other than the web server machine that provides the user interface. This program can easily be run remotely from another machine by simply using a Unix command line function.

3.1.1 Remote Procedure Call

A remote procedure call (RPC) works much like a local procedure call. In the local case, the caller places arguments to a procedure in a well specified location (such as a register). The calling procedure then transfers control to the procedure, and eventually the caller regains control. At this point, the results of the procedure are extracted from the specified location, and the caller continues with execution.

The remote procedure call is similar to a local procedure call. One thread of control moves through the following two processes: the callers’ process and a receiving process. The caller process first sends a call message to the server process and waits for a reply message. The call includes the procedure’s parameters, and the reply
Figure 3.1 Distributed Topology
This is the distributed manner in which the Ohio University Mailing List program runs. Macintoshes, workstations and PCs all run the Java GUI locally. The Java GUI talks to the Java Server with all its requests. The Java Server calls the RPC front end with certain arguments, and the RPC front end talks to the back end. The RPC back end server executes requests and sends a response back to the RPC front end. The RPC front end passes the response back to the Java Server, which then sends the response directly to the user Java GUI.
includes the procedure's results. Once the reply message is received, the results of
the procedure are extracted, and the callers' execution is resumed. RPC can use
many transport protocols, but this program uses TCP [Pos81], [Ste97]. TCP is a
reliable protocol that will help guarantee that each request will get to the server.
Other protocols may get the request to the server, but they cannot guarantee the
answer will get back. It is possible that if the answer gets lost, some protocols may
respond by making the request again. This is not the sort of behavior that should
occur; the server should get only one request per attempt to help eliminate ambiguity.

3.2 GUI interface

Since the advent of the Apple Macintosh, programmers have been more aware
of the importance of the user interface [Lau94]. A Graphical User Interface (GUI)
should be an important component of any program. A good GUI should enable a
user to use a program to its maximum potential [Bor94]. Java [GM95] was used to
create the front end GUI to the main program (see appendices B.2 and B.3).

The Java language is useful because it can run over Netscape; therefore, anyone can
download and use a Java program (applet) using Netscape. Java provides a powerful
addition to the tools that programmers have at their disposal. Java makes program-
ming easier because it is object-oriented and has automatic garbage collection. In
addition, because compiled Java code is architecture-neutral, Java applications are
ideal for a diverse environment like the Internet.

The OU mailer interface begins by prompting the user for his e-mail address (see
figure 3.2). Once the user enters his name and hits return, he is prompted for his
password (see figure A.1). The password he enters is checked against a password the
user has previously entered on a Unix system, which uses a program that is provided.
By requiring users to run this Unix password program first, it does the following three
This is the initial screen after loading the Ohio University Mailer program. At this point, the user inputs his e-mail address and hits return. After being prompted for, and entering, his password, the user is logged onto the program.

things: it gives users a safe place to set an initial password or to change a password, it uses Unix's user identification to verify the user’s identity, and it restricts users of this program to be users of a certain machine or domain.

The interface will not allow any other actions until the user is logged on. After the user has logged on and the front end has confirmed with the back end that the password is correct, the user may begin to use the program. Initially, the user has the following four options:

1. He may add himself to a public mailing list (see figure A.2).

2. He may remove himself from any list he is on (see figure A.3).
3. He may create a new mailing list.

4. He may modify a current mailing list that he already maintains.

![Image of creating a mailing list](image)

**Figure 3.3 Creating a Mailing List**
The user is prompted for the name of the mailing list he wishes to create.

When a user creates a mailing list, he is prompted for the name of the list that he wishes to create (see figure 3.3). At this point, the mailing list modification menu comes up (see figure 3.4).

From the modification menu, a user can change some of the functions of the list and the list membership. A maintainer may choose whether a mailing list is either join-able to the public or a private list. He can also choose if a list will only be mailable by its members or if a list will be moderated. A moderated list is one in which all e-mail sent to the list goes to the maintainer. It will not go to the rest of the list unless the maintainer approves it.

From the modification menu, a maintainer may also either add users to the mailing list or delete them. Choosing to delete a user from the list will bring up a menu of all list members (see figure A.4). The maintainer needs only to check a box beside the people he wants to remove and then press the "Remove these users" button. The last choice the maintainer has from this menu is to delete the whole list itself.
Figure 3.4 Modifying a Mailing List
After either creating a mailing list or picking a list to modify, this menu appears. From here the user can make the list public or private, make the list moderated, make the list mailable only by its members, add a user, remove a user, and delete the list.

If the user simply wants to modify an existing list he owns, he can click on the “Modify list” button. A list of all the mailing lists the user maintains will come up on the screen (see figure 3.5). The user needs to click on the list he wishes to modify, and then the program will bring up the modification menu for the selected list.

3.3 PERL E-mail Resender

When e-mail is sent to a mailing list, the entry in the /etc/aliases file tells the computer to run a PERL script (see appendix B.4). This PERL script was taken from MajorDomo release 1.93 and slightly modified. The script looks at the e-mail message sent to the mailing list and decides if the e-mail should be sent on to the list members. The script has an option to discard e-mail that does not meet the requirements (such as a submission by a non-member) and will do this if certain arguments are given to it at run time. The script can also forward e-mail to the list owner for approval. This
3.4 Other Considerations

One concern of network programming is always scalability. A program may work well for a single case or a single user, but the greater challenge is making a program work for many people at once. Using Java helps the scalability of the program because many clients can be run on individual computers and used independently of each other. If the OU mailer GUI had run on only one computer, the scalability of the program would have been limited by the number of GUI programs that computer could execute at one time. There is another scalability issue with the GUI at the moment. If there are numerous mailing lists or numerous members to a list, some of the GUI menus may grow too large for the screen to show at one time. In the GUI menu scalability case, the menu size is limited by the resolution of the monitor.

The second concern is concurrency. While many users can run clients, only one client can access the back end server at one time. If users could access and modify the mailing list data base concurrently, then the program might run into the problem...
of Readers and Writers concurrency [SG94]. RPC helps to take care of this problem. The RPC server for the back end will queue all requests and run those requests one at a time. By processing the requests serially, the server will not have a concurrency problem.
4. CONCLUSIONS

The OU mailer is effective at allowing users to join, create, modify, and remove themselves from mailing lists. Each mailing list has many options and properties to make the use of that list easier and more effective. A mailing list can be public so that anyone can join it, or the list can be private so that only the maintainer of the list can add people to it. A mailing list can be set so that only group members may send e-mail to that list; this adds more privacy to the mailing list. Lists created with this program can even be moderated so that all messages sent to the list go straight to the maintainer. Then, the maintainer can approve the e-mail to go to the rest of the list, or he can reject it.

The OU mailer also has a great deal of security built into it. There is a password scheme to prevent a person from posing as another user. The user even has to enter his password on a Unix system. This provides good identification and a safe place to keep the passwords. There are tradeoffs between having too much security and too little. This program has enough security built into it to make it difficult to fake an identity, but it is simple enough not to limit the scope of its users. Outside users will have difficulties discovering the members of private lists. This is because the mailing list program does not modify the /etc/aliases file directly. It adds a level of indirection telling the sendmail program to run a script that will send out the e-mails to the list. List members are stored in a file that is hidden from users. With the use of this file, it is impossible for someone to run the EXPN command on the mail port in order to find out the list members names.

The interface has been made with a GUI design. Previous mailing list programs have had complex interfaces that could confuse potential users. By making a friendly
GUI interface with help menus, the potential for confusion is much less. Writing the GUI interface in Java gave the interface some very nice enhancements. The interface will now run on any computer with a Java enabled web browser on it. Running over the World Wide Web makes the interface easy to load and to run, because downloading the program and running it is seamless to the user.

The OU mailer currently runs over two computers in the Internet Research Group lab; the front end runs on Jarok.cs.ohiou.edu, and the back end runs on Thrawn.cs.ohiou.edu. The GUI interface can be loaded at http://jarok.cs.ohiou.edu/~bwilson/oumailer/oumailer.html and needs a Java compliant Netscape to run. Before the GUI can be logged into, the user needs to set his password on Prime.cs.ohiou.edu by running the program /home/bwilson/passwdadd.

The OU mailer program has been used by my colleagues and myself for the last year to keep in touch with each other. This program is robust, secure, and easy to use. The Ohio University Mailer offers many features that a good mailing list needs. It offers public/private lists, moderated lists, and exclusive sending rights to those lists. With the password scheme, users can be confident that other users cannot pose as them. The password scheme also limits the number of users that can log onto the mailing list program and the number of users that can create mailing lists. Most importantly, the program is easy to use. With a limited number of user options, help menus, and helpful directions, the interface can be mastered in minutes.
BIBLIOGRAPHY
BIBLIOGRAPHY


APPENDIX
A. SCREEN SHOTS

This appendix contains more screen shots of the OU mailer program.

Figure A.1 Entering a Password
This is where the user enters his password for the OU mailer.

Figure A.2 Joining a Mailing List
This is a list of all the mailing lists that are public and therefore it is join-able to the user. By checking off the lists he wants to join and clicking the “Add me to these lists” button, the user will join all the lists he checked.
Figure A.3 Quitting a Mailing List
This is a list of all the mailing lists the user is a member of. By checking off the lists he wants to quit and clicking the “Remove me from these lists” button, the OU mailer will remove the user from all the lists he checked.

Figure A.4 Removing People from a Mailing List
This is a menu for a maintainer of a mailing list. It holds the names of every member of that list. By checking the users he wishes to remove and clicking the “Remove these people from the list” button, the users will be removed from that list.
B. SOURCE CODE

B.1 Back End Source Code

B.1.1 Mailer.c

This is source code for the back end file Mailer.c

/*
 * mailer.c - handles automatic updates of the mail alias file
 * The messages get piped into standard input by the delivermail
 * program.
 * This program has setgid(adm) to allow writing the alias files.
 *
 * Modified by Brian Wilson June, 1997
 * -Modified this program to fit into the Ohio University Mailer
 *
 * - added maintainence of the separate file used at Purdue U.
 *
 * Written by Bill Nowicki May 30, 1981
 * - Mailing list file maintenance added June 2, 1981
 * - Continuation line bug fixed July 10, 1981
 * - Maximum size checking added October 16, 1981
 * - Maintainers allowed to delete from lists January 28, 1982
 * - Interpret delete as remove and vice versa
 * depending on simple heuristics May 12, 1982
 * - Bug in Add and extra remove smartness November 1982
 * - Ignore "Please", upper case February 1983
 * - Fixed bug in public detection March 1983
 * - Fixed bug in testing only prefixes of reserved words
 * April 1983
 *
 * Wishlist:
 * - Check for simultaneous updates (lock database)
 * - Commands to destroy & modify mailing list data
 * (eg maintainers)
 * - Use "parse" from Eric's stuff, and pretty-print long
mailing lists along several lines.
- Convert strings with parity bit on into quoted strings.

Copyright (c) 1982 by William I. Nowicki

#include "mailer.h"

char *MkTemp();
char *GetL();
char *Get();
struct passwd *pw, *getpwnam(); /* some external procedures */
extern char *index();
extern FILE *popen();

extern int Debug; /* true if debugging */
extern FILE *logfd; /* log file pointer */
extern int Lock; /* lock file */
extern char *TempName;
extern char User[256], /* the user who sent the mail */
Subject[256], /* the subject of the mail */
Maint[256], /* maintainers for a mailing list */
Description[MAXSIZE], /* Human-readable description */
Date[256], /* date of creation for lists */
Public[256]; /* non-empty means anyone can add */

int html; /*This is the html flag */
int fLocalUser; /* true if invoked by local user */

typedef struct node {
    char string[256]; /* This is for looking up the list file dir*/
    struct node *next;
} node;

#define LOCALENTRY "localentry"/* just a marker in the list */
char *rgsbCheck[64] = {GROUP, HOME, EXTRA, SYSTEM, LASTNAME,
    LOCALENTRY, NULL};

DBM *dbm;

***** Passwdchk *****
/* takes a username, a passwd and returns true if the specified user has that passwd */
passwdchk(words)
    char **words;
{
char *passwd, *user, *value;
char name[100];
FILE *passfile;

user = *words++;
passwd = *words;

/*** I have to add a : after the name because that's how it appears in the PASSWORD file ****/

strcpy(name, user);
strcat(name, ":");

/* printf("user is %s and its %d chars\n", name, strlen(name)); */
if(!Isinlist(name, ":", "Passwd", Scan)){
  printf(":false password:\n");
  printf(" No password for %s defined, you need to run the
       Passwdadd program on ace.\n", user);
  return(-1);
}

/*** This uses user rather than name because GetL automatically
    appends a : to the end of the first variable ***/
if((value=GetL(user, "Passwd")) == NULL){
  printf(":false password:\n");
  /* Shouldn't get in here... unless there is some sort of reading
     problem */
  /* printf(" value = %s \n", value); */
  return(-1);
}

if(strcmp(value, passwd)){
  /* Does the given passwd match the actual passwd */
  printf(":false password:\n");
  return(-1);
}

printf(":true password:\n");
return(0);

}

/***** PasswdAdd **********/
passwdadd(words)
char **words;
{
    char *passwd;
    char line[MAXSIZE];
    FILE *passfile, *tmp;
    char name[100];
    char tmpfile[300];

    /***
    *** I have to add a : after the name becuase thats how it
    appears in the PASSWD file ****/
    strcpy(name, User);
    strcat(name,":");
    passwd = *words;

    if(Isinlist(name,"","Passwd", Scan)){
        /* Find and replace existing passwd */
        passfile = fopen(PASSWDFILE,"r+b");

        if(passfile == -1 || passfile == 0){
            printf("Could not open up the file: %s \n",PASSWDFILE);
            return(-1);
        }
        strcpy(tmpfile,PASSWDFILE);
        strcat(tmpfile,".tmp");
        tmp = fopen (tmpfile,"w");
        if(tmp == -1 || tmp == 0){
            printf("Could not open up the /tmp/tmpPASSWD for
                    writing \n");
            fflush(stdout);

            return(-1);
        }

        while(fgets(line, MAXSIZE, passfile)){
            if(FoldedEQ(name,line,strlen(name))){
                fprintf(tmp,"%s: %s\n",User,passwd);

                fflush(stdout);
            } else
                fprintf(tmp,line);
    }

    /* FIND THE ENTRY AND REPLACE IT */
fclose(passfile);
fclose(tmp);
if(rename(tmpfile,PASSWDFILE) != 0){
    printf("Problems renaming the Password file. Please 
report to %s \n",MAILWIZARD);
    return(1);
} return(0);
}
passfile = fopen(PASSWDFILE,"ab");
if(passfile == -1 || passfile== 0){
    printf("Could not open up the file: %s \n",PASSWDFILE);
    return(-1);
}

fprintf(passfile,"%s: %s\n",User,passwd);
fclose(passfile);

return(0);
} /***** TXT to HTML **************/
txt_to_html( how , var1, var2,var3)
char *how;
char *var1;
char *var2;
char *var3;
/***** Actually used for list public (html) *****/
{
    printf( "<LI><INPUT TYPE="checkbox" Name="%c ",10);
    printf("%s", var1 );
    printf("\" VALUE="" );
    printf("%s ",var1 );
    printf("\">");
    printf( how, var1,var2,var3 );
    printf(" <P>%c\n",10);
}
Mod_list(words)
char **words;
{
    /* This Function should sake some listname and the options
* that need to be added to that list (In the aliases file)
*/

char *list,*maint,*passwd;
int iflag,aflag;
char listname[256];
char passfilename[256],command[256];
FILE *aliasfile, *tmpfile, *passwdfile;
struct hostent *hname;
char hostname[64];
char buff[MAXSIZE];

list= *words++;
if (FoldedEQ(*words,"with",4)){
    *words++;
    maint=*words++;
}

if (Debug) fprintf(stderr," In mod list! List=%s maint=%s\n",
                     list,maint);

if(FoldedEQ(*words,"Ion",3))
    iflag=1;
else
    iflag=0;

if (Debug) fprintf(stderr,"Options are %s and %s \n",
                     *words++,*words);

if(FoldedEQ(*words,"Aon",3))
    aflag=1;
else
    aflag=0;
if (Debug) fprintf(stderr," So those opts are %d and %d \n",
                     iflag,aflag);

if(aflag){
    *words++;
passwd=*words;

    strcpy(passfilename,ALIASDIR2);
    strcat(passfilename,"passwd/");
    strcat(passfilename,list);
    strcat(passfilename,".pass");
    passwdfile= fopen(passfilename,"w");
    fprintf(passwdfile,"%s\n",passwd);
fclose(passwdfile);
}

if(IsMailingList(list,Maint,Description,Public,Date)){
   if(!IsOn(maint,Maint,Scan) && !IsMailWizard(maint)){
      printf("You are not the maintainer of this list, nor are
      the the Wizard\n");
      return(1);
   }
}
else{
   printf(" List does not Exist \n");
   return(1);
}

strcpy(listname,list);
strcat(listname,":");

strcpy(command, ALIASDIR);
strcat(command,"aliases\0");
aliasfile = fopen(command,"r");
tmpfile = fopen("/tmp/temp","w");

while(fgets(buff,MAXSIZE,aliasfile))
{
   if(!strncmp(listname,buff,strlen(listname))){
      gethostname(hostname, sizeof(hostname));
      hname= gethostbyname(hostname);

      if(aflag && iflag)
         fprintf(tmpfile,"%s: \"%swrapper resend
            -1 %s -h %s %s -A -a %s -s -outgoing\"%n","list,RESENDDIR,
            list,hostname,ALIASDIR2,list
           ,passfilename,list);
      else if(aflag && !iflag)
         fprintf(tmpfile,"%s: \"%swrapper
            resend -1 %s -h %s -A -a %s
            ")
      
      
   }
}
else if (!aflag && !flag)  
  fprintf(tmpfile, "%s: \"|\%swrapper 
  resend -1 %s -h%ls -I %s%s  
  %s-outgoing\"\n", list,  
  RESENDDIR, list, hostname  
  , ALIASDIR2, list, list);

else if (!aflag && !flag)  
  fprintf(tmpfile, "%s: \"|\%swrapper resend  
  -1 %s -h %s %s-outgoing \"\n"  
  , list, RESENDDIR  
  , list, hostname, list);

}  
else  
{
  if (fputs(buf, tmpfile) == EOF)  
    printf("Tmp file full?\n");
}

fclose(aliasfile);
fclose(tmpfile);

strcpy(command, RESENDDIR);
strcat(command, "wrapper\0");
if (fork() == 0) {
  execl(command, "wrapper", "doaliases", NULL);
  exit(1); /* just incase of a runaway child */
}

return(0);
}

/******************Probe-list***************************/
Probe_list(user, funct)
char *user;
char *funct;
{

  DIR *dirp;
  struct dirent *direntp;
  struct node *head, *current, *temp;


head = (node*)malloc(sizeof(node));
/* direntp = (struct dirent*)malloc(sizeof(struct dirent)); */
strcpy(head->string,"HEADPTR");

current=head;

dirp.opendir(ALIASDIR2);
if(dirp==NULL)
    if (Debug) fprintf(stderr,"Got a null")
while ( (direntp = readdir( dirp )) != NULL )
{
    if((strcmp(direntp->d_name, ".\0")
        &&(strcmp(direntp->d_name, "..\0")
        &&(strcmp(direntp->d_name, "passwd\0")})
        temp = (node*)malloc(sizeof(node));
        temp->next = NULL;
        current->next = temp;
        current=temp;
        strcpy(current->string,direntp->d_name);
}
closedir( dirp );
if(head->next!=NULL)
    head=head->next; /*If the first node isnt null, move head to the 1st node*/
if(funct=="WHICHLIST") /*Show whihclists the user is on */
{
    printf(":Remove from:\n");
    while (head != NULL) { /* Set head to point to last node */
        if ( Isinlist( user, head->string,"Aliasdir2", Scan ) )
            printf("%s \n",head->string);
        temp=head;
        head = head->next;
        free(temp); /* Hopefully this frees the malloced memory */
    }
}
if(funct=="LPU"
{
    /** print out all lists, but only make pub lists joinable. 
     * Give the maintainer names of the private lists.
    */
    printf(":Add to:\n");
    while (head != NULL) {

if(IsMailingList(head->string,Maint,Description,Public,Date) 
&& (!strcmp(Public,"public",6)))
printf("publ:%s\n",head->string);

temp=head;
head = head->next;
free(temp); /* Hopefully this frees the malloced memory */
}

} /* end LPUB */
/*** I NEED TO FREE NODES!!!!!!! ***/
printf(":End:\n");/*this should help my front end know when its*/
/* stoped getting list data */
}
******** End Probe_list ******************/

/*
 * InitCheckList() - initialize alias lists to check
 */
InitCheckList()
{
    int i;
    DIR *d;
    #if defined(__svr4__)
        struct dirent *dp;
    #else
        struct direct *dp;
    #endif
    struct stat filestat;
    FILE *f;
    char sbName[100], *pch, *pchMe;

    for (i = 0; i < 64; i++)
        if (rgsbCheck[i] == NULL)
            break;
    /* make sure local aliases are first on list */
pchMe = rgsbCheck[i] = (char *)malloc(strlen(sysname)+1);
strcpy(rgsbCheck[i], sysname);
if ((pch = (char *)index(rgsbCheck[i], '.')) != NULL)
    *pch = '\0';
rgsbCheck[++i] = NULL;

d = opendir(ALIASDIR);
if (d == NULL) {
    return(-1);
}
while (dp = readdir(d)) {
    if (dp->d_ino == 0)
        continue;
    strcpy(sbName, ALIASDIR);
    strcat(sbName, dp->d_name);
    if (stat(sbName, &filestat) == 0) {
        if (((strncmp(dp->d_name, "local.", 6) == 0) &
             ((char *)index(&dp->d_name[6], ')') == NULL) &
             (strcmp(&dp->d_name[6], pchMe) != 0)) {
            /* a local entry, add to list */
            rgsbCheck[i] = (char *)malloc(strlen(&dp->d_name[6])+1);
            strcpy(rgsbCheck[i], &dp->d_name[6]);
            rgsbCheck[++i] = NULL;
        }
    }
}
closedir(d);

Help()
{
    PrintCommands();
    return( 0 );
}            /* Help */

/*
 * resolve turns "username" into a complete DNS address.
 *
char *
resolve_new_broken(username)
    char *username;
{
    static char retbuf[100];
    char hostname[64];
    char *hn1, *hn2, *hn3;
    char *dname;

    gethostbyname(hostname, sizeof(hostname));

#ifdef OHIOU
      /*
      */
#endif
* make sure we got a full domain name */
if (index(hostname, '.') == NULL) {
    strcat(hostname, DOMAIN_SUFFIX);
}
#endif OHIOU

/*
* break hostname apart: xxx.cs.ohiou.edu
*/
hn1 = (char *) index(hostname, '.') + 1; /* cs.ohiou.edu */
hn2 = (char *) index(hn1, '.') + 1; /* ohiou.edu */
hn3 = (char *) index(hn2, '.') + 1; /* edu */

dname = (char *) index(username, '@');
if (dname == NULL) {
    /* no "@", assume local machine */
    sprintf(retbuf, "%s@%s", username, hostname);
} else if (index(dname, '.') == NULL) {
    /* just a single name, like 'ace', so tack on suffix */
    sprintf(retbuf, "%s.%s", username, hn1);
} else {
    /* it's of the form "user@host.foo" */
    sprintf(retbuf, "%s.%s", username, hn2);
}

fprintf(stderr, "Resolving '%s' to '%s'\n", username, retbuf);

return(retbuf);

/*
* resolve will compare the "user@host-domain" address with
* the local hostname. Parts of the user address are stripped
* off if they are being send to a local machine
*/
resolve(username)
    char     *username;
{
    char     hostname[64];
    char     *dm1, *dm2, *hn1, *hn2, *hn3;
    int      last, slen;
gethostname(hostname, sizeof(hostname));

#ifdef OHIOU
    /*
     * make sure we got a full domain name
     */
    if (index(hostname,'.') == NULL) {
        strcat(hostname,DOMAIN_SUFFIX);
    }
#endif OHIOU

    /* break hostname apart: xxx.cs.ohiou.edu */
    hn1 = (char *) index(hostname, '.') + 1;  /* cs.ohiou.edu */
    hn2 = (char *) index(hn1, '.') + 1;  /* ohiou.edu */
    hn3 = (char *) index(hn2, '.') + 1;  /* edu */

    if ((dm1 = (char *) index(username, '@')) != NULL) {
        last = dm1 - username;
        dm1++;  /* first part of address after '@' */
        while (dm1 != NULL) {
            if (!strncmp(dm1, hn2, hn3 - hn2)) {
                dm2 = NULL;
                slen = strlen(dm1);
            } else if (((dm2 = (char *) index(dm1, '.')) != NULL) {
                dm2++;
                slen = dm2 - dm1;
            } else
                slen = strlen(dm1);

            if (strncmp(dm1, hn2, hn3 - hn2 - 1) 
                && strncmp(dm1, hn1, hn2 - hn1 - 1) ||
                !strncmp(dm1, hn1, hn2 - hn1 - 1) && dm2 != NULL &&
                strncmp(dm2, hn2, hn3 - hn2 - 1))
                last = last + slen;
            dm1 = dm2;  /* next part of address */
        }
        if (username[last] == '.' || username[last] == '@')
            username[last] = '\0';
        else
            username[last+1] = '\0';
return(0);
}

MakeAlias( words )
char **words;
{
    /*
       * Add the alias for the user
       */
    char *alias, *user, *value, **save = words;
    char host[20];
    int i, fLocal;

    alias = *words++;

    if (index(alias,'@') != NULL) {
        printf("alias is for making aliases but it looks like you\n");
        printf("want to forward mail so forwarding will be done.\n");
        return(Forward(save));
    }
    if ( FoldedEQ( *words, "for", 999) ||
         FoldedEQ( *words, "as", 999) )
        words++;

    if ( *words )
        user = *words++;
    else user = User;
    if ( FoldedEQ( user, "me", 999) ) user = User;
    /*resolve(user);*/

    if ( FoldedEQ( *words, "at", 999) )
        words++;

    if ( *words ) {
        strcpy(host,*words++); /* this is a host at which user is aliased */
    } else {
        strcpy(host,"");
    }

    if ( !FoldedEQ( user, User, 999) && !IsMailWizard(User) )
{  
    printf( "You can only add aliases for yourself.\n");  
    return( 1 );  
}

/* check if alias already exists */
fLocal = 0;  
for (i = 0; rgsbCheck[i] != NULL; i++) {  
    if (strcmp(rgsbCheck[i], LOCALENTRY) == 0) {  
        fLocal = 1;  /* all entries after  
            this are local */  
            continue;  
    }
    if ((value = GetL(alias, rgsbCheck[i])) != NULL) {  
        if (fLocal)  
            printf( "The alias %s is already in use on %s  
                , and maps to:\n", alias, rgsbCheck[i]);  
        else  
            printf( "The alias %s is already in use, and maps  
                to:\n", alias );  
        PrettyPrint(1, value, " ");
        return( 1 );
    }
}

if (strcmp(host,"")) == 0) {
    if ( Update( alias, user, EXTRA, Add) )
        return( 1 );
} else {
    if ( Update( alias, user, host, Add) )
        return ( 1 );
}

printf("%s successfully added as an alias for %s.\n",  
    alias, user );
if (Debug) fprintf(stderr,"MakeAlias:%s successfully  
        added as an alias for %s.\n", alias  
    , user );

    return( ShouldBeNull(words) );  
}  
/* MakeAlias */

AddUser( words )
char **words;
{
    /*
     * Add a user to a mailing list
     */
    char *list, *user, *value, *realmaint;
    int publicFlag = 0;

    realmaint = User; /*the person running the program is the assumed maint*/
    if (words[1])
        user = *words++;
    else user = User;

    if (FoldedEQ(user, "me", 999))
        user = User;

    if (FoldedEQ(user, "public", 6))
        publicFlag++;

    if (FoldedEQ(*words, "to", 3))
        words++;

    list = *words++;
    /***** Added to take the with Maint arguments *****/
    if (FoldedEQ(*words++, "with", 4)) {
        printf("Another word taken\n");
        realmaint = *words;
    }
    strcpy(Maint, "");

    /*resolve(user);*/

    /*resolve(user);*/

    if (IsMailingList(list, Maint, Description, Public, Date)) {
        if (publicFlag) { /* attempt to change list to public */
            if (!IsOn(realmaint, Maint, Scan) && !IsMailWizard(User)) {
                printf("You can only make a list public if you are a maintainer.\n");
                if (Debug) fprintf(stderr,"You can only change a list you are on\n");
                return (1);
            }
        }
    }
else

   if (Debug) fprintf(stderr,"Calling changelist\n")
   return(ChangeList(list, Maint, "publist", Date,
                      Description, Add));

} else {

   if ( !FoldedEQ( Public, "public", 6)
        && !IsOn( realmaint, Maint, Scan )
        && !IsMailWizard(User) ) {
      printf( "%s is not a public mailing
               list. Only the following
               maintainers can
               add to it.\n", list);
      PrettyPrint(1, Maint, " ");
      return(1);
   }

}                                        
/*******************************************/
/* might have to change GROUP to LISTFILE */
if ( ( value=GetL(list,LISTFILE) )==NULL )
   { printf( "There is no mailing list
            named %s.\n", list);
     return(1);
   }

if ( strlen(value)+strlen(user)+strlen(list) + 2
     >= MAXSIZE )
   { printf( "Sorry, adding %s to %s would make
            the
            list too long.\n", user, list);
     return(1);
   }

/********* None of this seems important right
       now ****/
printf("--------------------------------------\n");

if ( Isinlist( user, list,"Aliasdir2", Scan ) )
   { printf( "%s is already on the list %s.\n" 
            , user, list);
     return(1);
   }
printf("AFTER ISONLIST \n");
printf("user=%s User=%s hope this helps! \n",user,User);

if ( !FoldedEQ(user,realmaint,999) &&
!IsOn( realmaint, Maint, Scan )
&& !IsMailWizard(User) )
{
    printf("You can only add yourself to that mailing list.\n");
    return( 1 );
}

/*if (*value)
    strcat( value, "\n");// This was screwing up my insertion */
strcpy(value,user); /*strcat( value, user ); */
/*listfile?*/
if ( UpdateList( list, value, GROUP, Add )
  return( 1 );
printf("%s successfully added to the %s mailing list.\n",user, list );
if (Debug) fprintf(stderr,"AddUser:%s successfully added to the %s mailing list
 .\n", user, list );

return( ShouldBeNull(words) );
}

/* AddUser */

PlusLots( words )
char **words;
{
    /*
    * add many members (but not maintainers) to a mailing list
    */
    char *list, *user, *value;
    char newuser[64];
list = *words++;
strcpy( Maint, "" );

if ( IsMailingList( list, Maint, Description, Public, Date ) ) {
    if ( !FoldedEQ( Public, "public", 6 ) &&
         !IsOn( User, Maint, Scan )
         && !IsMailWizard(User) ) {
        printf( "%s is not a public mailing list,
                so only %s can add to it.\n", list, Maint );
        return( 1 );
    }
}

if ( !IsMailingList( list, Maint, Description, Public, Date ) ) {
    if ( !FoldedEQ( Public, "public", 6 ) &&
         !IsOn( User, Maint, Scan )
         && !IsMailWizard(User) ) {
        printf( "%s is not a public mailing list,
                so only %s can add to it.\n", list, Maint );
        return( 1 );
    }
}

while( (user = *words++ ) != NULL )
{
    /*resolve(user);*/
    if ( Isinlist( user, list,"Aliasdir2",
                    Scan ) )
    {
        printf( "%s is already on the list %s
                .\n", user, list);
        continue;
    }
    else
    {
        printf("users=%s!\n",user);
        strcpy(value,user);
        if ( Updatelist( list, value, GROUP, Add ) )
            return( 1 );
printf("Members successfully added to the %s mailing list.\n",list);
}

if (Debug) fprintf(stderr,"Members successfully added to the %s mailing list.\n",list);

return( ShouldBeNull(words) );
} else
    printf("Pluslots: %s is not a mailinglist.\n",list);
    /* PlusLots */

Create( words )
char **words;
{
    /*
    * create a mailing list with the given members
    */
    char *list, *user, *value;
    char string[MAXSIZE];
    char mstring[MAXSIZE];
    char line[256];
    int publicFlag = 0, i, fLocal;

    if (Debug)
    {
        char **p = words;
        fprintf( stderr, "Create: Words are: ");
        while (*p) fprintf( stderr, "%s", *p++ );
        fprintf( stderr, "\n" );
    }

    if (!fLocalUser) {
        if (Debug) fprintf(stderr, "Not a local user, cannot create.\n") ;
        printf("You are not an Ohio University CS Dept. user, and hence cannot create a mailing list.\n");
    }
/* gobble up the description */
while (!feof(stdin) && !FoldedEQ(line,"--------", 8))
{
    if (GetNextLine(line)== (int)NULL) break;
}
return(1);

if ( FoldedEQ( *words, "public", 6 )
{
    words++;
    publicFlag++;
}

list = *words++;

if ( FoldedEQ( *words, "public", 6 )
{
    words++;
    publicFlag++;
}

if ( FoldedEQ( *words, "with", 4 )
words++;

/* check if alias already exists */
fLocal = 0;
for (i = 0; rgsbCheck[i] != NULL; i++) {
    if (strcmp(rgsbCheck[i], LOCALENTRY) == 0) {
        fLocal = 1; /* all entries after this are local */
        continue;
    }
    if ((value = GetL(list, rgsbCheck[i])) != NULL) {
        if (fLocal)
            printf( "The alias %s is already in use on 
            %s, and maps 
            to:
", list, rgsbCheck[i]);
        else
            printf( "The alias %s is already in use, 
            and maps to:
", list );
        PrettyPrint(1, value, " ");
    } /* gobble up the description */
while (!feof(stdin) && !FoldedEQ(line,"--------", 8))
{
    if (GetNextLine(line)== (int)NULL) break;
}
return( 1 );
}

strcpy( string, "" );
strcpy( mstring, "X" );
while ( user = *words++ ) {
    if ( FoldedEQ( "maintainer", user, 10 ) ) {
      /* allow maintainer or maintainers */
      /* start maintainers list */
      strcpy(mstring, "" );
      break;
    }
    if ( FoldedEQ( user, "public", 6 ) ) {
      publicFlag++;
      continue;
    }
    if ( FoldedEQ( user, "me", 999 ) ) user = User;
    /*resolve(user);*/

    if (*string)
        strcat( string, "," );
    strcat( string, user );
}

if (mstring[0] != '\0') {/* no special maintainers list */
    strcat(mstring, User);/*just use the user as maintainer*/
}
else {
    while ( user = *words++ ) {
        if ( FoldedEQ( user, "public", 6 ) ) {
            /* do nothing */
            continue;
        }
        if ( FoldedEQ( user, "me", 999 ) ) user = User;
        /*resolve(user);*/
        if (*mstring)
            strcat( mstring, "," );
        strcat( mstring, user );
    }
}

if ( strlen(mstring)==0 ) {
    printf( "There must be at least one maintainer.\n" );
/* gobble up the description */
while (!feof(stdin) && !FoldedEQ(line,"--------", 8)) {
    if (GetNextLine(line) == (int)NULL) break;
} return(1);

if (!IsMailingList(list, Maint, Description, Public, Date)) {
    if (Debug) fprintf(stderr,"Create: not found on mailing lists file.\n");
    /*
     * read the description and put it into the
     * mailing list file.
     */
    strcpy( Description, "" );
    strcpy( line,"" );
    while (!feof(stdin) && !FoldedEQ(line,"--------", 8)) {
        if (GetNextLine(line) == (int)NULL) break;
        if (Debug) fprintf(stderr, "Create: reading description: %s\n", line);
        if (!FoldedEQ(line,"--------", 8)) {
            strcat( Description, "\t" );
            strcat( Description, line );
            strcat( Description, "\n" );
        }
    }
    Updatealiasfile(list,string,"ADD","NO OPS");
    CreateMailingList(list, string, Description, publicFlag ? "Public" : "", (char *)arpadate() );
} else {
    /* not sure why program should ever get here --- cew */
    if (Debug) fprintf(stderr,"Create: Found %s on maintainers file, maintained by %s\n",list,Maint);
    if (strcmp(Description,"") != 0))
        printf("\tDescription:%s\n",Description);
    printf("You have tried to create a list that already exists!
    Please try another name. \n");
    exit(1);
    /* added by bri, this used to let people try to
    create a list that already existed, it wouldn't
let them change the list around, but it DID
append their email to the list */
/* Swallow any description (if no description follows,
* command
* will be swallowed by mistake)
*/
while (GetNextLine(line) != (int)NULL
   && !FoldedEQ(line,"--------", 8)) {
    }

} /*This line puts the creator into the mailing list file,
   they are now a member Of that list*/
if ( UpdateList( list, string, GROUP, Add ) ) return( 1 );
printf("Successfully created the %s mailing list.
", list );
if (Debug) fprintf(stderr,"Create:Successfully created
   the %s mailing list.
", list );

return( 0 );

} /* Create */

UpdateAliasFile(list,owner,function,option)
char *list;
char *owner;
char *function;
char *option;
{
    char hostname[64];
    FILE *aliasfile, *tmpfile;
    struct hostent *hname;
    char buff[MAXSIZE];
    char delname[1000],delname1[1000],delname2[1000];
    char command[256];
    int bricount ;

if(!strcmp(function,"ADD")) /*add function*/
{
    /************ make alias file here willy! *****/
    if (Debug) fprintf(stderr,"In updatealiasfile ADD
        got list=%s and owner=%s
",list,owner);
        gethostname(hostname, sizeof(hostname));
        hname= gethostbyname(hostname);
        strcpy(command,ALIASDIR);        
        strcat(command,"aliases\0");
        aliasfile = fopen(command,"r");


if (aliasfile == -1 || aliasfile == 0) {
    printf("There is a problem opening the Alias file\n");
    return(-1);
}
tmpfile = fopen("/tmp/temp","w");
if(tmpfile == -1 || tmpfile == 0) {
    printf("Could not open up the /tmp/temp file \n");
    return(-1);
}
fprintf(tmpfile,"%s: \"|\%swrapper resend -l %s -h %s
%s-outgoing\"\n",list, RESENNDDIR, list, hostname, list);
fprintf(tmpfile,"%s-outgoing: :include:%s\n",list,
ALIASDIR2, list);
fprintf(tmpfile,"owner-%s: %s\n",list,owner);
while(fgets(buff, MAXSIZE, aliasfile)) {
    if(fputs(buff, tmpfile) == EOF)
        printf("Is the Tmp file full?\n");
}
fclose(aliasfile);
fclose(tmpfile);
strcpy(command, RESENNDDIR);
strcat(command, "wrapper\0");
if(fork() == 0) {
    execl(command, "wrapper", "doaliases", NULL);
    exit(1); /* just incase of a runaway child */
}
/*****************************************************/
else if(!strcmp(function, "REMOVE"))
{
    if (Debug) fprintf(stderr,"IN updatealiasfile REMOVE \n");
    bricount = 0;
    strcpy(command, ALIASDIR);
    strcat(command, "aliases\0");
    aliasfile = fopen(command, "r");
    tmpfile = fopen("/tmp/temp","w");
    strcpy(delname, list);
    strcat(delname, ":");
    strcpy(delname1, list);
    strcat(delname1, "-outgoing:");
    strcpy(delname2, "owner-");
strcat(delname2,list);
strcat(delname2,":")

while(fgets(buff,MAXSIZE,aliasfile)!= NULL )
{
    bricount++;
    if(!strncmp(delname,buff,strlen(delname)))
        sleep(1); /* Do nothing*/
    else if(!strncmp(delname1,buff,strlen(delname1)))
        sleep(1); /* do nothing */
    else if(!strncmp(delname2,buff,strlen(delname2)))
        sleep(1); /*You guessed it, do nothing */
    else
    {
        if(fputs(buff,tmpfile)==EOF)
            printf("Tmp file full?\n");
    }
}
if (Debug) fprintf(stderr,"Whats this buff say? %s bri
count is %d \n",buff,bricount);
fclose(aliasfile);
fclose(tmpfile);
/*/tmpfile and /tmp/temp need to be changed to use .h defualts */
/* need to be root to write to aliasfile */
if(Debug) fprintf(stderr,"Before exec\n");

strcpy(command,RESENDDIR);

strcat(command,"wrapper\0");
if(fork()==0){
    exec1(command,"wrapper","doaliases",NULL);
    exit(1); /* JUST incase of a runaway child */
}
}
else
{
    printf("Update can either ADD or REMOVE from a list, not
%ss\n",function);
}
Delete( words )
char **words;
{
    /*
    * Command to take a user out of a mailing list.
    */
    char *list, *user, *names, *value,**save=words,*realmaint;
    /*char *Maint,*Description,*Public,*Date;*/
    int publicFlag = 0;

    realmaint=User;
    if (words[0])
        user = *words++;
    else {
        user = User;
        list=**words;
        //*** I moved this from Deletelist ***/
        if ( !IsMailWizard(User) && ( !IsMailingList( list, Maint,Description, Public, Date ) || !IsOn( User, Maint, Scan ) ) )
        {
            printf(" User:%s
",User);
            printf("user:%s
",user);
            printf("list:%s
",list);
            printf("Maint:%s
",Maint);
            printf("desc:%s
",Description);
            printf("Public:%s
",Public);
            printf("Date:%s
",Date);
            printf("You can only delete a list for
            which you
            are a maintainer.
\n");
            if (Debug) fprintf( stderr, "You can only
            delete a list
            which you are on.
\n");
            return( 1 );
        }
    }
    return(DeleteList(*words)); /* Delete a list */
}

if ( FoldedEQ( user, "me", 999 ) )
    user = User;

if ( FoldedEQ( user, "public", 6 ) )
publicFlag++;

/***Take the name of the list owner, if given******/

if(FoldedEQ( *words, "with", 4 )
{
    *words++;
    list=user; /* the list name has been stored as
                 the username */

    if ( !IsMailingList( list, Maint,
                        Description, Public, Date )
        || IsOn( *words, Maint, Scan ) )
    {
        if (Debug) fprintf(stderr,"Taking given owner
            name,
            deleteing %s
            list);
        return(DeleteList(list));
    }
    else{ /* this user (*words) doesent own this list */
        printf("You can only Delete a list which you own
            %s
            isnt the same as %s",*words,Maint);
        return(1);
    }
}

/**************************************************************************/

if ( FoldedEQ( *words, "from", 4 )
/* Delete a person from a list */
    words++;
else {
    printf("Proper syntax is: delete NAME from LIST.\n");
    return(1);
}

/*resolve(user);*/

list = *words++;

if(FoldedEQ( *words, "with", 4 )
{
    *words++;
    realmaint=*words;
}

if (FoldedEQ( list, User, 999 ))
{
    printf( "Delete applies to mailing lists, 
            but it looks like\n" );
    printf( "you want to remove an alias for 
            yourself.\n\n" );

    printf( "Please try: delete NAME from LIST\n" );
    printf( " or remove NAME for USER [at HOST]\n" );
    return(1);
}

if ( (names = GetL(list,LISTFILE)) == NULL )
{
    printf( "There is no such mailing list as %s.\n", list);
    return( 1 );
}

if ( IsMailingList( list, Maint, Description, Public, 
                    Date ) )
{
    if ( publicFlag ) { /* make list non-public */
        if ( !IsOn( realmaint, Maint, Scan ) &&
            !IsMailWizard(User)) {
            printf("You can only delete public from a 
                    list that you are a maintainer.\n");
            if (Debug) fprintf(stderr,"You can 
                    only delete public 
                    from a list that you 
                    are 
                    a maintainer.\n");
            return (1);
        }
    }
    else {
        if (Debug) fprintf(stderr,"Calling 
                    changelist\n");
        return(ChangeList(list, Maint, 
                    \0", Date, 
                    Description, Add));
    }
}

} else {

    if ( !FoldedEQ(user,realmaint,999)
        && !IsMailWizard(User) &&
        ( !IsMailingList( list, Maint, Description, 
                        Public,Date ) )
        ||
        !IsOn( realmaint, Maint, Scan ) )
    {
printf( "You can only delete yourself from
    that mailing list.\n" );
return( 1 );
}

if ( !Isinlist( user, list, "Aliasdir2", Eliminate ) )
{
    printf( "%s was not on the %s mailing list.
        \n", user, list );
    if (Debug) fprintf(stderr, "Delete: %s was not on the %s mailing list.
        \n", user, list );
    return( 1 );
}
else{
    /* lets just use update to erase this entry */

        if ( Updatelist( list, user, GROUP, Eliminate ) )
        return( 1 );
        printf("%s deleted from the %s mailing list.\n", user, list );

} /* end else*/
}

if (Debug) fprintf(stderr,"Delete:%s deleted from the %s mailing list.\n", user, list );

return( ShouldBeNull(words) );
} else
    printf("%s is not a mailing list.\n",list);
    /* Delete */

DeleteList( list )
char *list;
{
    /*
        * Command to delete a list entirely
        */
char *names;
char delname[100], delname1[100];
if (Debug) fprintf(stderr,"DeleteList:%s called by user
%s\n",list, User);

if (!fLocalUser) {
    if (Debug) fprintf(stderr, "Not a local user, cannot
delete.\n");
    printf("You are not an Ohio University CS Dept. user,
    and
    hence cannot delete a mailing list.\n");
    return(1);
}
/* Changed GROUP to LISTFILE */
if ( (names = GetL(list,LISTFILE)) == NULL )
{
    printf("There is no such mailing list as %s.\n",
    list);
    if (Debug) fprintf( stderr, "There is no such
mailing
    list as %s.\n", list);
    return(1);
}
if (Debug) fprintf(stderr,"DeleteList: names is %s\n",
    names);

Updatealiasfile(list,"NONE","REMOVE","no opts");
if ( (Update( list, names, LISTFILE, Eliminate ))
    return(1);
strcpy(delname,ALIASDIR2);
strcat(delname,list);
if(remove(delname))  /*Removes the list-file */
    fprintf(stderr,"There was a problem deleting %s \n"
    ,delname);
strcpy(delname1,delname);
strcat(delname,".passwd");
strcat(delname1,".config");
remove(delname1); /* This removes the config file if
    one exists */
remove(delname); /*this removes the password file if
    its there*/
if (Debug) fprintf(stderr,"DeleteList: now calling
    ChangeList\n");
if ( ChangeList( list, "\0", "\0", "\0", "\0", 
    Eliminate ) ) return(1);
printf("List %s deleted.\n", list);
if (Debug) fprintf(stderr,"DeleteList: List %s deleted.\n", list);

return( 0 );

} /* DeleteList */

Forward( words )
char **words;
{

/*
 * Like an alias, but the other way 'round
 */
char *alias, *user, *previous;

if (words[1])
    user = *words++;
else user = User;

if ( FoldedEQ( user, "me", 999 ) ) user = User;

if ( FoldedEQ( *words, "to", 3 ) )
    words++;
alias = *words++;

if ( !FoldedEQ(user,User,999) && !IsMailWizard(User) )
{
    printf( "You can only add aliases for yourself.\n");
    return( 1 );
}

if (gethostbyname(alias) != NULL) {
    printf("The address you gave is a host. This address\n is not valid.\n");
    printf("You should use the following command:\n");
    printf(" forward %s to %s@%s\n", user, user, alias);
    return(1);
}

previous = GetL(user,HOME);

if ( Update( user, alias, HOME, Add ) ) return( 1 );
printf("Mail sent to %s will be forwarded to %s.\n", user, alias);
if (Debug) fprintf(stderr,"Forward:Mail sent to %s will be forwarded

if (previous)
    printf("It used to be forwarded to %s.\n", previous);

/* added by bri */
return( ShouldBeNull(words) );
    /* Forward */

Keep( words )
char **words;
{
    /*
    * eliminate forwarding for this user
    */
    char *host, *user, alias[MAXSIZE];

    if (*words)
        user = *words++;
    else user = User;

    if ( FoldedEQ( user, "me", 999 ) ) user = User;

    if (Debug) fprintf(stderr, "Keep: user = %s ",user);

    if ( FoldedEQ( *words, "at", 3 ) )
        words++;

    if ( *words )
        host = *words++;
    else
        host = sysname;

    if (Debug) fprintf(stderr,"host: %s
",host);

    if ( !FoldedEQ(user,User,999) &amp; !IsMailWizard(User) )
    {
        printf( "You can only change the forwarding for yourself.\n");
        return( 1 );
    }

    /* resolve(user); */
    strcpy (alias, "");
    strcat (alias, user);
    strcat (alias, "@");
    strcat (alias, host);
if ( Update( user, alias, HOME, Add ) ) return( 1 );
printf("Mail to %s@%s will not be forwarded.\n", user, host);
if (Debug) fprintf(stderr, "Keep:Mail to %s@%s will not be forwarded.\n", user, host);

/*added by bri */
return( ShouldBeNull(words) );
} /* Keep */

Remove( words )
char **words;
{
    /*
     * eliminate an alias for this user
     */
    char *alias, *user, *value, **save=words;
    char host[20];
    alias = *words++;
    /*resolve(alias);*/

    if ( FoldedEQ( *words, "from", 4 ) )
    {
        printf( "Remove is for aliases, but it looks like you wanted to\n" );
        printf( "delete %s from a mailing list.\n", alias );
        return( Delete( save ) );
    }
    if ( FoldedEQ( *words, "for", 999)
         || FoldedEQ( *words, "as",999) )
        words++;
    if ( *words )
        user = *words++;
    else user = User;

    if ( FoldedEQ( *words, "at", 999 )
        words++;
    if ( *words )
        strcpy(host,*words++);
    else
strcpy(host,"" );
if ( IsMailingList( user, Maint, Description, Public, Date ) )
{
    printf( "Remove is for aliases, but it looks like you wanted to\n" );
    printf( "delete %s from a mailing list.\n", alias );
    return( Delete( save ) );
}

if ( FoldedEQ( user, "me", 999 ) ) user = User; /*resolve(user);*/

if ( !FoldedEQ(user,User,999) && !IsMailWizard(User) )
{
    printf( "You can only remove aliases for yourself.\n" );
    return( 1 );
}

if ( (value=Get(alias))==NULL )
    if ( (value=Get(user) ) && FoldedEQ(value,alias, strlen(alias) ) )
        printf( "The Remove command is for aliases, but it looks like\n" );
        printf( "you want to cancel forwarding to %s.\n" , value);
        if (*words==NULL) words--;
        return( Keep(words) );
    printf( "The alias %s does not exist.\n", alias );
    return( 1 );
}

if ( !FoldedEQ(value, user, strlen(user)) )
    printf( "The alias %s maps into %s, not you.\n", alias, value );
    return( 1 );

if ( strcmp(host,"" ) == 0 )
    if ( Update( alias, (char *) NULL,(char *) EXTRA, Add ) )
        return( 1 );
else {
    if ( Update( alias, (char *) NULL, (char *) host, Add ) )
return( 1 );
}
printf("%s successfully removed as an alias for %s.\n", alias, user);

return( ShouldBeNull(words) ); /* Remove */

ListAll( words ) /* I didn’t really even use this */
char **words;
{
    /*
     * list out all the aliases in order
     */
    #ifndef ALIAS_MAP
    datum key;
    FILE *pipe = popen( "sort", "w" );
    #endif
    if(html==0){ /* No HTML */
        printf( "The following is a list of all the aliases.\n" );
        fflush( stdout );
    }
    else{
    }

    #ifdef ALIAS_MAP
    system("/bin/ypcat -k mail.aliases | /bin/sed -e 's/ /:/’ | /bin/sort");
    #else
    for (key = dbm_firstkey(dbm); key.dptr; key = dbm_nextkey(dbm))
    {
        if (strncmp(key.dptr,"YP_LAST_MODIFIED", 16)==0
            || strncmp(key.dptr, "YP_MASTER", 9)==0)
            continue;

    if(html==0) /* No HTML */
        fprintf( pipe, "%s:%s\n", key.dptr, Get( key.dptr ) );
    else
        txt_to_html("%s:%s\n",key.dptr,Get(key.dptr));
    }
pclose( pipe );
#endif

return( ShouldBeNull( words ) );

LPub( words )
char **words;
{
    char *list;
    /*
     * Calls probe_list with the list pub option
     * /

    Probe_list("NO-User","LPUB");

    return( ShouldBeNull( words ) );
}    /* LPub */

static char *WhichListFiles[] = {
    GROUPFILE,
    EXTRAFILE,
    SYSTEMFILE,
    LASTNAMEFILE,
    ALIASFILE
};

WhichList( words )
char **words;
{
    /*
     * list all lists for which the user is a member
     * /
    char *user;

    if (*words)
        user = *words++;
    else
user = User;
if (FoldedEQ(user, "me", 999))
  user = User;

Probe_list(user,"WHICHLIST");
return( ShouldBeNull( words ) );
} /* WhichList */

WhichMaint( words )
char **words;
{
  /*
   * list all lists for which the user is a maintainer
   */
  char *user;
  char sbBuf[MAXSIZE];
  FILE *listFile = fopen( LISTFILE, "r" );
  char line[MAXSIZE], *value, *end;

  if (*words)
    user = *words++;
  else
    user = User;
  if (FoldedEQ(user, "me", 999))
    user = User;
  /*resolve(user>;*/

  sbBuf[0] = '\0';

  if (listFile==NULL) {
    fprintf(stderr, "Mailing list file not found, please
      contact %s\n",MAILWIZARD);
    if (Debug) fprintf(stderr,"IsMailingList: listfile
      not found ");
    return(0);
  }

  while ( fgets( line, MAXSIZE, listFile ) ) {
    if ((value = (char *)index(line, ':')) ! = NULL) {
      *value++ = '\0';
      if ((end = (char *)index(value, ':')) == NULL)
        *value = '\0';
      else
        *end = '\0'; /* list is between value and end */
if (IsOn(user, value, Scan)) {
    if (sbBuf[0] != '\0')
        strcat(sbBuf, ",");
    strcat(sbBuf, line);
}
}
strcat(sbBuf,"\0"); /* Not technically correct to have a comma at the end*/
    /* but it helps with parsing the output!*/
fclose(listFile);
if (sbBuf[0] == '\0')
    printf( "The user %s is not a maintainer of any lists. \n", user);
else {
    printf( "The user %s is a maintainer of the following lists:\n", user);
    PrettyPrint(1, sbBuf, " ");
    printf(":End of WhichMaint:");
}
return( ShouldBeNull( words ) );
} /* WhichMaint */

List( words )
char **words;
{
    /* List the value of the given aliases and/or mailing lists */
    char *value, *avalue, *filename;
    char sbShort[SYSNAMELEN], line[1024], *pch;
    int i, fLocal;
    FILE *aliasFile;

    while ( *words )
    {
        if ( IsMailingList( *words, Maint, Description, Public, Date ) )
        {
            PrettyPrint(0, *words, "List: ");
            PrettyPrint(1, Maint, "Maintained By: ");
            PrettyPrint(0, strlen(Public) ? "Yes": "No", "Public: ");
            PrettyPrint(0, Description, "Description: ");
printf("Members ");

/* Insert willy print */
filename = (char *)malloc(128);
strcpy(filename, ALIASDIR2);
strcat(filename, *words);
if ((aliasFile = fopen(filename, "r")) == NULL) {
    printf("LIST: Could not open the file %s for reading.
", filename);
    if (Debug) fprintf(stderr, "List: Could not open the alias file %s for reading.
", filename);
    return(1);
}

while(1) {
    if (fgets(line, MAXSIZE, aliasFile) == NULL)
        break;
    else
        printf("%s", line);
}
printf(":Maintainer Delete:");
words++;
}
return(0);

Update( keyString, contentString, whichfile, option)
char *keyString, *contentString, *whichfile;
enum Option option;
{
    /*
     * This routine actually updates the alias database.
     * We edit the source file, then the database is rebuilt by
     * a cron job at night.
     *
     * Returns non-zero on any errors.
     */
    FILE *aliasFile, *temp;
    char line[MAXSIZE], start[MAXSIZE];
char outs[MAXSIZE];
char realname[MAXSIZE];
char *filename;
int size = strlen(keyString) + 1, continuation = 0;
int delete;

if (index(keyString, ':')) {
    printf( "Aliases and list names cannot contain a colon\n"");
    return (1);
}

if (Debug)
    if (option == Eliminate)
        fprintf(stderr,"Update: Eliminate\n");
    else if (option == Add)
        fprintf(stderr,"Update: Add\n");

if (Debug) fprintf(stderr,"Update: TempName:%s
\n
TempName\n",TempName);

if (Debug) fprintf(stderr,"\nUpdate: keyString :%s\n\ntcontentString: %s\n\nwhichfile: %s\n",keyString,
contentString,
whichfile);

if (strcmp(whichfile,GROUP) == 0) {
    filename = GROUPFILE;
} else if (strcmp(whichfile,HOME) == 0) {
    filename = HOMEFILE;
} else if (strcmp(whichfile,EXTRA) == 0) {
    filename = EXTRAFILE;
} else if (strcmp(whichfile,SYSTEM) == 0) {
    filename = SYSTEMFILE;
} else {
    filename = (char *)malloc(128); /* get some space */
    strcat(filename, whichfile);
}

if (Debug) fprintf(stderr,"Update: filename is %s\n",filename);

if ((aliasFile=fopen( filename, "r")) == NULL) {

if (Debug) printf(stderr, "Update: Could not
open the alias
file %s for reading.\n", filename);
    return( 1 );
}
if ( (temp=fopen( TempName, "w"))==NULL)
{
    printf("Could not open the temporary file. 
Please report to 
%s.\n", MAILWIZARD);
    if (Debug) printf(stderr, "Update: Could not open the 
temporary file. \n");
    return( 1 );
}
strcpy( start, keyString );
strcat( start, ":" );

while ( fgets( line, MAXSIZE, aliasFile ) )
{
    delete = 0;
    /*
    * if we found the desired entry,
    * replace it and continuation lines with the new
    * contents,
    * but just copy all other lines.
    */
    if (Debug) printf(stderr,"In update, is %s=%s for 
\%d\n",start,line,strlen(start));
    if ( FoldedEQ (start,line, strlen(start)))
    {
        if (Debug) printf(stderr,"in update, DID find a match\n");
        if (Debug) printf(stderr,"Update: found a match %s\n",start);
        if (option == Add) {
            strncpy(realname,line,strlen(start));
            realname[strlen(start)] = '\0';

            if ( contentString )
                fprintf (temp, "%s%s\n", 
                                     realname, contentString);
            contentString = NULL;
            continuation = 1;
continue;
}
if (option == Eliminate)
    delete = 1;
}
if (continuation && isspace(line[0])) continue;
continuation = 0;
if (delete == 0)
    if (fputs(line, temp) == EOF) goto failed;
else
    if (Debug) fprintf(stderr, "Don’t print line %s
", line);
}
if ((contentString) && (option != Eliminate))
    if (fprintf(temp, "%s%s
", start, contentString) == EOF) {
        failed:
        printf("Problems updating database (full file system?). Please report to %s
", MAILWIZARD);
        unlink(TempName);
        return(1);
    }
fclose(temp);
fclose(aliasFile);

/*
 * This is the tricky part. Use rename to rename the
 * temp file.
 */

if (rename(TempName, filename) != 0) {
    printf("Problems renaming database. Please report to %s
", MAILWIZARD);
    /*unlink(TempName);*/
    return(1);
}
free(filename);
return(0);
}  /* Update */

Updatelist(keyString, contentString, whichfile, option)
char *keyString, *contentString, *whichfile;
enum Option option;
{
    /* modified for list updates
    * This routine actually updates the alias database.
    * We edit the source file, then the database is rebuilt by
    * a cron job at night.
    *
    * Returns non-zero on any errors.
    */
    FILE *aliasFile, *temp;

    char line[MAXSIZE], start[MAXSIZE];
    char outs[MAXSIZE], line2[MAXSIZE];
    char realname[MAXSIZE];
    char *filename;
    int size = strlen(keyString) + 1, continuation = 0;
    int delete;
    int ret, getnewline, i;
    char ch;

    if (index(keyString, ':')) {
        printf("Aliases and list names cannot contain
        a colon\n");
        return (1);
    }

    if (Debug)
        if (option == Eliminate)
            fprintf(stderr,"Update: Eliminate\n");
        else if (option == Add)
            fprintf(stderr,"Update: Add\n");

    if (Debug) fprintf(stderr,"Update: TempName:%s\n======
FROMNAME==\n",TempName);
    if (Debug) fprintf(stderr,"\nUpdate: keyString:%s\n" contentString:
    "%s\n" whichfile:"%s\n",keyString,
    contentString, whichfile);

    filename = (char *)malloc(128);
    strcpy(filename, ALIASDIR2);
strcat(filename, keyString);

if (Debug) fprintf(stderr, "Update: filename is %s\n", filename);

if ((aliasFile=fopen(filename, "r")) == NULL) {
    if (Debug) fprintf(stderr, "Update: Could not open the alias file %s for reading.\n", filename);
    return(1);
}

if ((temp=fopen(TempName, "w")) == NULL)
{
    printf("Could not open the temporary file. Please report to %s.\n", MAILWIZARD);
    if (Debug) fprintf(stderr, "Update: Could not open the temporary file. \n");
    return(1);
}

/* I fear I may have to come back ot this point and make sure the unix lines arent Growing too long */

ret=1;
getnewline = 1;
/*
 * if we found the desired entry, replace it and continuation lines with the new contents,
 * but just copy all other lines.
 */

if (Debug) fprintf(stderr, "Update: found a match %s\n", start);
if (option == Add) {
    /* go to botom of list*/
while(1)
{
    if(fgets( line, MAXSIZE,
            aliasFile )==NULL)
    {
        break;
    }
    else
        fprintf (temp, "%s", line);
}

if ( contentString )
    fprintf (temp, "%s\n",
             contentString);
contentString = NULL;
continuation = 1;
/*continue;*/
delete=1;
} /*end add*/

continuation = 0;
if (option == Eliminate)
    while(1)
    {
        if(fgets( line, MAXSIZE,
                  aliasFile )==NULL)
            {
                break;
            }
        else if((size=strlen( line ) -1)
                 && (!FoldedEQ(line,
                           contentString,size)))
            {
                fprintf (temp, "%s", line);
                /*take out the newline*/
            }
else  if (Debug) fprintf(stderr, "Don't print line %s\n", line);
  */*/
if (!contentString && (option != Eliminate))
  if (fprintf(temp, "%s%s\n", start, contentString) == EOF) {
    printf("Problems updating database(full file system?). Please report to %s.\n", MAILWIZARD);
    unlink(TempName);
    return(1);
  }
free(filename);
fclose(temp);
fclose(aliasFile);
/*
 * This is the tricky part. Use rename to rename the temp file.
 */
if (rename(TempName, filename) != 0) {
  printf("Problems renaming database. Please report to %s.\n", MAILWIZARD);
  unlink(TempName);
  return(1);
}
return(0);
  /* Updatelist */

ChangeList( list, maint, public, date, desc, option )
char *list, *maint, *public, *date, *desc;
enum Option option;
{
  /*
   * changes the information to the mailing list file
   */
  FILE *listFile = fopen(LISTFILE, "rw");
  FILE *mailinglists;
  FILE *temp;
  char *tempName;
  char line[MAXSIZE];
char *matchlist;
int added;
char *p;
int i;

if (index(list, ':')) {
    printf( "Aliases and list names cannot contain a colon\n");
    return (1);
}
tempname = MkTemp (TEMPLATEMLIST);
matchlist = (char *) malloc (MAXSIZE);

strncpy(matchlist, list, strlen(list));
matchlist[strlen(list)] = ':';
matchlist[strlen(list)+1] = '\0';

if (Debug) fprintf( stderr, "ChangeList:list:%s
    maint:%s
    public:%s
    list,maint,public);
if (Debug) fprintf( stderr, "desc %s
    tt
desc)
if (Debug) fprintf( stderr, "tempname %s
    tempname); 
if ( (temp=fopen( tempname, "w"))==NULL) {
    printf( "Could not open the temporary file.
        Please report to %s.
        MAILWIZARD); 
    if (Debug) fprintf(stderr, "ChangeList: Could not open the
        temporary file.\n");
    return(1);
}
if (listFile==NULL) {
    printf( "Unable to open %s.\n", LISTFILE );
    fprintf(stderr, "Unable to open %s.\n", LISTFILE );
    return(1);
}

added = 0;
while ( fgets(line, MAXSIZE, listFile) ) {
    if ((index(line, ':') == 0)) /* description line */
        if ((option == Eliminate) && (added == 1))
            continue; /* just skip over the match */
    else if ((added == 1)
&& (option != Eliminate) &&
    (strlen(desc) != 0))
    continue;
    /* we have a new description, skip */
    if (fprintf( temp, "%s", line) == EOF)
        goto failed;  /* go ahead and print */
    continue;
}
if ((added == 1) && (option != Eliminate) &&
    (strlen(desc) != 0))
    if (fprintf(temp,"%s", desc) == EOF) goto failed;
    added = 0;
if ( FoldedEQ (matchlist, line, strlen(matchlist)) ) {
    if (Debug) fprintf(stderr,"Found a match
     %s
",line);
    if (option != Eliminate)
        if (fprintf( temp,"%s:%s:%s:%s
",list,
                     maint,public,date ) == EOF)
            goto failed;
        added = 1;
    }
if (added == 0) {
    if (fprintf( temp, "%s",line) == EOF)
        goto failed;
}
/* need to add description at the bottom of the file */
if ((added == 1) && (option != Eliminate) && (strlen(desc)
    !=0))
    if (fprintf( temp, "%s", desc) == EOF) {
        failed:
        printf( "Problems updating database (full file
        system?). Please
        report to %s.
        MAILWIZARD);
        unlink(tempname);
        return( 1 );
    }
fclose(listFile);
fclose(temp);
/*
 * This is the tricky part. 
 */
if (rename( tempname, LISTFILE ) != 0) {
printf("Problems renaming control database. 
Please report to %s\n", MAILWIZARD);
unlink( tempname );
return( 1 );
}
if ((option != Eliminate) &&
( IsMailingList( list, Maint, Description, Public,
Date ))) {
printf("\nCurrent status of the mailing list:\n");
PrettyPrint(0, list, "List: ");
PrettyPrint(1, Maint, "Maintained By: ");
PrettyPrint(0, strlen(Public)?"Yes":"No",
"Public: ");
PrettyPrint(0, Description, "Description: ");
}
return( 0 );
/* ChangeList */

CreateMailingList( list, string, description, public, date )
char *list, *string, *description, *public, *date;
{
    /*
    * appends the information to the mailing list file
    */
    char aliasdir[512];
    int num;
    FILE *listFile = fopen( LISTFILE, "a" );
    FILE *listname ;

    /* a new aliases file should be created for Domo use */

    aliasdir[0]= '\O';
    num = strcat(aliasdir, ALIASDIR2);
    num = strcat(aliasdir, list);
    listname = fopen( aliasdir, "w" );

    if (index(list, ':')) {
        printf("Aliases and list names cannot contain a colon\n")
        return ( 1 );
    }
if (Debug) fprintf(stderr, "CreateMailingList: list:%s, string:%s, description:%s, public:%s, date:%s
", list, string, description, public, date);

if (listFile==NULL)
{
    printf("Unable to append to %s.\n", LISTFILE);
    return(1);
}

fprintf(listFile, "%s%s%s%s\n", list, string, public, date);

/*fprintf(listname, "Test \n10;*/
fclose(listname);
fclose(listFile);
return(0);
    /* CreateMailingList */

/*******************************************************************************/
/* setup things and get ready */
/*******************************************************************************/
Initialize(fInter)
int fInter; /* is program called in interactive mode? */
{

    if (Debug) fprintf(stderr, "Initialize: Ready to open %s\n", LOCKFILE);

    if ((Lock = open(LOCKFILE,O_CREAT|O_WRONLY,444)) < 0) {
        if (Debug) fprintf(stderr,"Initialize: problem opening %s\n",LOCKFILE);
        perror("Initialize");
        return(1);
    }

    if (Debug) fprintf(stderr,"Initialize: opened the %s successfully\n",LOCKFILE);
if (Debug) fprintf(stderr,"Initialize: about to flock %s\n",LOCKFILE);

if (fInter) {    /* not willing to wait long */
    #if defined(hpux) || defined(__svr4__)
        if (((lockf(Lock, F_TLOCK, 0) < 0) && ((errno != EAGAIN) &&
            (errno != EACCES)))
            #else
                if (((flock(Lock, LOCK_EX|LOCK_NB) < 0) && (errno
                    != EWOULDBLOCK))
                    #endif
            if (Debug) fprintf(stderr,"Initialize: problem
                with flock %s\n",LOCKFILE);
                perror("Initialize");
                return(1);
        }
    else if (errno == EWOULDBLOCK) {
        sleep(5); /* wait a few seconds and try again */
        #if defined(hpux) || defined(__svr4__)
            if (lockf(Lock, F_TLOCK, 0) < 0) {
                #else
                    if (flock(Lock, LOCK_EX|LOCK_NB) < 0) {
                        #endif
                            if (Debug) fprintf(stderr,"Initialize: Program in use.\n");
                            printf("Database currently in use, please try again later.\n");
                            return(1);
                    }
            }
    }
    else {    /* willing to wait */
        #if defined(hpux) || defined(__svr4__)
            if (lockf(Lock, F_TLOCK, 0) < 0) {
                #else
                    if (flock(Lock, LOCK_EX) < 0) {
                        #endif
                            if (Debug) fprintf(stderr,"Initialize: problem
                                with flock %s\n",LOCKFILE);
                                perror("Initialize");
                                return(1);
                    }
            }
        }
    }

    if (Debug)
        fprintf(stderr, "Initialize: Ready to open database %s\n",}
#ifdef OLD
    if (dbminit(ALIASFILE) < 0)
        exit(EX_OSFILE);
#else
    dbm = dbm_open(ALIASFILE, O_RDONLY, 0700);
    if (dbm == NULL)
        exit(EX_OSFILE);
#endif

#ifndef FourPointOne
    gethostname(sysname, SYSNAMELEN);
#endif FourPointOne

        /* attempt to open the log file, if not there,
         * that’s ok, we just won’t
         * write to it.
         */
    if (Debug) fprintf(stderr, "Initialize: Ready to open
    logfile\n");
    logfd = fopen(LOGFILE, "a");

        /* create name of the temporary file */
    if (Debug) fprintf(stderr, "Initialize: making temp file\n");
    TempName= MkTemp( TEMPLATEALIAS );
    if (Debug) fprintf(stderr, "Temp file name %s\n", TempName);
    fclose(logfd);    /* added by bri */
    return(0);

}            /* Initialize */

/*
 * trap all trappable signals to ensure the lock file goes away
 */
stopme()
{
    if (Debug) fprintf(stderr,"** signal trap *******\n");
    #if defined(hpux) || defined(__svr4__)
        lockf(Lock, F_ULOCK, 0);
    #else
        flock(Lock, LOCK_UN);
    #endif
    exit( 1 );
main(argc, argv)
char **argv;
{
    char *pch;

    fLocalUser = 0;
    if ((argc > 1) && (strcmp(argv[1], "-d") == 0)) {
        /*
         * decode options, mostly for debugging
         */
        Debug = 1;
        argv++;
        argc--;
    }

    /******************************************************************************
     * ADDED BY BRI .................................................................
     */
    html = 0;
    if ((argc > 1) && (strcmp(argv[1], "-html") == 0)) {
        /*
         * decode options, mostly for debugging
         */
        html = 1;
        argv++;
        argc--;
    }
    html = 1; /* Note this will always give html code *****/
    /** that was residue from an old idea*/
    if (argc < 2) {
        fprintf(stderr, "No user given in the command line.\n");
        exit(1);
    }

    (void) umask( 002 );
    if (setgid( getegid() ) != 0) {
        printf("Problems setting proper permissions.
            Please report to %s.\n", MAILWIZARD);
        exit(1);
    }

    if (Debug) fprintf( stderr, "main: Ready to set
            signals\n" );

    signal(SIGHUP,SIG_IGN);
signal(SIGINT,SIG_IGN);
signal(SIGQUIT,SIG_IGN);
signal(SIGILL,SIG_IGN);
signal(SIGTRAP,SIG_IGN);
signal(SIGFPE,SIG_IGN);
signal(SIGBUS,SIG_IGN);
signal(SIGTERM,SIG_IGN);

if (ReadFromLine(User, argv[1]))
    exit(EX_OSERR);

/*resolve (User);
*/

fLocalUser = 1;  /* assume local user */
for (pch = User; *pch != '\0'; pch++) {
    if (!(*pch >= 'a' && *pch <= 'z') &&
        !(*pch >= 'A' && *pch <= 'Z')) {
        fLocalUser = 0;  /* not from a local user */
    }
    break;
}

if (Debug) fprintf(stderr,"User: %s %s a local user\n", User, fLocalUser ? "is" : "is not");

InitCheckList();
if (DoCommand()) {
    printf("Send mail to %s for more help, or send me a help command.\n", MAILWIZARD);
}
fclose(stdout);
#if defined(hpux) || defined(__svr4__)
    lockf(Lock, F_ULOCK, 0);
#else
    flock(Lock, LOCK_UN);
#endif

exit(EX_OK);  /* main */

nextfield(string,chunk)
char *string;
char *chunk;
{
    int i = 0;
}
```c
strcpy(chunk, string);
do {
    if (*chunk == ':') {
        *chunk = '\0';
        return(i);
    }
    i++;
} while (*chunk++);
return((int)NULL);

Maintainers( words )
char **words;
{
    /*
     * Change the maintainers of a list
     */
    char *list, *user;
    char string [MAXSIZE];

    user= User;
    list = *words++;
    if(FoldedEQ(*words, "with", 4)) {
        *words++;
        user=*words++;
    }

    if (Debug) fprintf(stderr, "Maintainers: list %s\n", list);

    /* must make sure IsMailingList is called first */
    if (( !IsMailingList( list, Maint, Description, Public, Date )
        || !IsOn( user, Maint, Scan ) ) && !IsMailWizard(user)) {
        printf("You must be a maintainer to change the
maintainers.\n");
        if (Debug) fprintf(stderr, "You must be a maintainer to
change the maintainers.\n");
        return( 1 );
    }

    strcpy( string,"" );
    while ( user = *words++ )
    {
        if ( FoldedEQ( user, "me", 999 ) ) user = User;
    }
```
/*resolve(user);*/
if (*string)
strcat( string, "," );
strcat( string, user );
}

if ( strlen(string)==0 )
{
    printf( "There must be at least one maintainer.\n" );
    return( 1 );
}
if (Debug) fprintf(stderr,"string of maintainers for list %s is %s\n",list,string);
if (Debug) fprintf(stderr,"Calling ChangeList\n");
return (ChangeList(list,string,Public, Date, Description, Add));

Describe( words )
char **words;
{
    /*
     * Change the description for a list
     */
    char *list,*user;
    char line[256];

    if(words[1])
    {
        list=*words++;
        if(FoldedEQ(*words,"with",4))
        {
            printf(" In if in describe \n");
            *words++;
            user=(char*)malloc(40);
            /*space for 40 chars for user name */
            user=*words;
        }
    } else
    {
        list = *words++;
        user=User;
    }
    /* must make sure IsMailingList is called first */
    if ( ( !IsMailingList( list, Maint, Description, Public, Date )

(!IsOn(user, Maint, Scan) && !IsMailWizard(user)) {
    printf( "You can only describe a list for which you are a maintainer.\n");
    if (Debug) fprintf(stderr, "You can only describe a list which you are on.\n");
    /* chew up description */
    while (!feof(stdin) && !FoldedEQ(line,"--------", 8)) {
        if (GetNextLine(line) == (int)NULL) break;
    }
    return(1);
}
if (Debug) fprintf(stderr, "Changing description of list %s\n",list);
/*
 * read the description and put it into the mailing list file.
 */
strcpy( Description, "" );
strcpy(line,"" );
while (!feof(stdin) && !FoldedEQ(line,"--------", 8)) {
    if (GetNextLine(line) == (int)NULL) break;
    if (Debug) fprintf(stderr, "Create: reading description: %s\n",line);
    if (!FoldedEQ(line,"--------", 8)) {
        strcat( Description, "\t" );
        strcat( Description, line );
        strcat( Description, "\n" );
    }
}
if (Debug) fprintf(stderr, "Description is %s\n", Description);
return(ChangeList(list,Maint,Public, Date, Description, Add));
}

char *
MkTemp(pattern)
    char *pattern;
{
    char *ret;
    char *buf;
    char *mktemp();

    buf = (char *) malloc(strlen(pattern) + 2);
struct syntax {
    char *sbCmd;            /* command name */
    int cMinWord;           /* min number of words for command */
    int cMaxWord;           /* max number (-1 for no max) */
    char *sbSyntax;         /* syntax for command */
    char *sbDescribe;       /* description for command */
} rgCmd[] = {
    "add", 4, 6,
    "add NAME to LIST                ",
    "Add the name to the mailing list.",

    "add", 4, 4,
    "add public to LIST              ",
    "Make the mailing list public.",

    "alias", 4, 6,
    "alias NAME for USER [at HOST]   ",
    "Make the alias for the user.",

    "lpub", 1, 1,
    "lpub                          ",
    "Print out all the public mailing lists.",

    "mod", 4, 7,
    "mod                           ",
    "Makes modifications to a list.",

    "all", 1, 1,
    "all                          ",
    "Print out all the aliases and mailing lists.",

    "create", 4, -1,
};
"create LIST with NAME,.. [maintainers NAME,..]",
"\nCreates the mailing list with the list of
names. The creator is the default maintainer.",

"delete", 2, 2,
"delete LIST
", "Delete the mailing list.",

"delete", 4, 4,
"delete NAME from LIST
", "Delete the name from the mailing list.",

"delete", 4, 6,
"delete NAME from LIST
", "Delete the name from the mailing list.",

"delete", 4, 4,
"delete public from LIST
", "Make the mailing list private.",

"describe", 2, 4,
"describe LIST
", "Give a description for the list.",

"forward", 4, 4,
"forward USER to ADDRESS
", "Forward mail, e.g. forward joe to joe@ace",

"help", 1, 1,
"help
", "Print this command summary.",

"keep", 4, 4,
"keep USER at HOST
", "Don't do any forwarding.",

"list", 2, -1,
"list NAME,.. 
", "List information about the given names.",

"maintainers", 3, -1,
"maintainers LIST NAME,.. 
", "Specify the maintainers of the list.",

"pluslots", 3, -1,
"pluslots LIST NAME,.. 
",
"Add many users to the existing list.",

"remove", 4, 6,
"remove NAME for USER [at HOST] ",
"Remove the alias for the user.",

"passwdadd", 2, 2,
"passwdadd, I DONT WANT THIS HERE ",
"Kill THIS EVENTUALLY ".

"passwdchk", 3, 3,
"passwdchk ",
"WORK DAMNIT! ".

"whichlist", 2, 2,
"whichlist NAME ",
"Print lists and aliases the user is on."

"whichmaint", 2, 2,
"whichmaint NAME ",
"Print lists the user is a maintainer of."

NULL, 0, 0, NULL, NULL

char sbMsg[512];

/*
 * CheckSyntax(sbBuf) - check basic syntax of the command line.
 * This routine will not catch everything, but will catch a lot.
 */
char *CheckSyntax(sbBuf)
char *sbBuf;
{
    int cWord, i, fMatch;

    cWord = CountWords(sbBuf);
    strcpy(sbMsg, "Invalid syntax, mailer expects:\n");
    fMatch = 0;
    for (i = 0; rgCmd[i].sbCmd != NULL; i++) {
        if (strncmp(sbBuf, rgCmd[i].sbCmd, strlen(rgCmd[i].sbCmd))
            == 0) {
            if ((cWord >= rgCmd[i].cMinWord) &&
                ((rgCmd[i].cMaxWord == -1)
                 || (cWord <= rgCmd[i].cMaxWord)))
return(NULL);      /* success */

else {
    strcat(sbMsg, "\t");
    strcat(sbMsg, rgCmd[i].sbSyntax);
    strcat(sbMsg, "\n");
}
    fMatch = 1;
}

if (!fMatch)
    sprintf(sbMsg, "Unknown command:\n\t%s\n", sbBuf);
return(sbMsg);

COUNTWORDS(sbBuf) - count the words (space or tab separated)
* in the buffer
*/
COUNTWORDS(sbBuf)
char *sbBuf;
{
    char *pch;
    int cWord = 0;

    pch = sbBuf;
    while (*pch != '\0') {
        /* eat up space */
        while ((*pch == ' ') || (*pch == '\t'))
            pch++;
        if (*pch != '\0')
            cWord++;
        /* eat up word */
        while (!(*pch == '\0' || *pch == ' ') && (pch != '\t'))
            pch++;
    }
    return(cWord);
}

PrintCommands()
{
    int i;

    printf("The current commands are: ([]’s denote optional, caps need values)\n");
    for (i = 0; rgCmd[i].sbCmd != NULL; i++)
        printf("%s%s\n", rgCmd[i].sbSyntax, rgCmd[i].sbDescribe);
B.1.3  Util.c

This is source code for the back end file Util.c

#include "mailer.h"

extern int Debug;  /* true if debugging */
struct passwd *pw, *getpwnam();  /* some external procedures */
extern FILE *logfd;  /* log file pointer */
extern char *index();
extern char User[256],  /* the user who sent the mail */
Subject[256],  /* the subject of the mail */
Maint[256],  /* maintainers for a mailing list */
Description[MAXSIZE],  /* Human-readable description */
Date[256],  /* date of creation for lists */
Public[256];  /* non-empty means anyone can add */
char templine[256];  /* input line */
int fCacheLine = 0;  /* is input line saved? */

ReadFromLine( from, user )
char *from, *user;
{
    /*
     * read and parse the "From" line that sendmail appends
     * to appease /bin/mail. This determines the sender.
     * Returns zero if everything went OK, 1 if it bombed out.
     */
    register char *p;

    if (fgets(templine, 256, stdin)==NULL) return(1);
    if (((p = index(templine, '\n')) != NULL)
        *p = '\0';
    if (Debug) fprintf(stderr,"ReadFromLine: read: %s\n",
        templine);

    if (strncmp( templine, "From ", 5) ||
        (p=index(templine+5, ', '))!=NULL) {
        /* not a valid From line so assume it is a command */
        fCacheLine = 1;
        strcpy(from, user);
    }
if ( Initialize(1) )  /* moved from main() */
    exit( EX_OSERR );
}
else {
    *p = 0;
    strcpy( from, templine+5);  
    /* check and make sure the from person is valid */
    if (strcmp(from, user) != 0) { 
        /* hmm, not the same */
        if (strcmp(user, SENDMAILDEFAULT) != 0) { 
            if (logfd != NULL) {
                fprintf(logfd, "You claim to be \\
                %s, but really \\
                are %s\n.", from, user);
            }
            printf("You claim to be %s, but really \\
                are %s\n.", 
                from, user);
            exit(1);
        }
    }
}
if ( Initialize(0) )  /* moved from main() */
    exit( EX_OSERR );  
/* code moved from main()---will send output to mail process */
if ( WriteResponse( User ) )
    exit( EX_OSERR );
    printf( "Response from %s Mail alias handler:\n
      , sysname );
}

if (Debug) fprintf( stderr, "ReadFromLine: Got request 
      from %s\n", from);
return(0);
}

WriteResponse( user )
char *user;
{
    /*
     * Writes back a response message to the indicated user
     * the message will be available as the standard output
     * stream.
     * Adds the start of the Arpanet RFC733 header.
     * Also returns nonzero on anomalies.
     */
int fd[2];

if ( pipe(fd) < 0 ) return(1);
if ( fork() == 0 )
{

dup2( fd[0], 0 );
close( fd[1] );
execl( SENDMAIL, "- (Responding)", "-eq", "-f",
mailer",user, 0 );
printf( "Could not do the exec of %s!!!\n", SENDMAIL );
if (Debug) fprintf(stderr, "Could not do the exec
of %s\n", SENDMAIL );
exit(1);
}

dup2( fd[1], 1 );
close( fd[0] );
if (Debug) fprintf( stderr, "WriteResponse: Sending
response back to %s\n", user);
printf( "Date: %s\nFrom: mailer\nTo: %s\n", arpadate(), user);
if (Debug) fprintf(stderr, "WriteResponse:Date: %s\nFrom:
mailer\nTo: %s\n", arpadate(), user);
return(0);
}

DoCommand()
{

/*
 * Perform the command for the user, after breaking
 * it up into words.
 * First we skip Arpa headers, if any.
 */

static char line[MAXSIZE];
static char *words[ 32 ];
char *linep;
char *pchMsg, *CheckSyntax();

strcpy( Subject, "" );
GetNextLine(line);
if (Debug) fprintf(stderr, "DoCommand: got: %s\n", line);

if (Debug) fprintf(stderr, "DoCommand:Response from %s Mail alias handler: \n\n", sysname);
if ((pchMsg = CheckSyntax(line)) != NULL) {
    printf("%s", pchMsg);
} else {
    IntoWords( line, words );

    if (Execute( words ) ) {
        if (Debug) fprintf(stderr, "DoCommand:Attempting to continue\n");
    }
}

while (((GetNextLine(line) != (int)NULL) && (strlen(line) != (int)NULL) && !FoldedEQ( line, "--------", 8)) {
    if (Debug) fprintf(stderr, "DoCommand: got: %s\n", line);
    printf("\n-----\n");
    /* separate output for multiple commands */
    if ((pchMsg = CheckSyntax(line)) != NULL) {
        printf("%s", pchMsg);
        continue;
    }

    IntoWords( line, words);
    if (Execute(words)) {
        if (Debug) fprintf(stderr, "DoCommand:Attempting to continue\n");
    }
}
return(0);
/* DoCommand */

static char LegalChars[] = "%@-/!._+:";

IntoWords( line, words )
register char *line, **words;
{
    /*
    * Lexical scanner.
    * Break the line up into a list of words.
    * Ignore punctuation and white space between words.
    */
* (Except for leading slashes for file names!!)
* /
static char table[1024];
register char *tablePointer = table;

while ( *line && isalnum( *line ) && *line != '/' &&
      *line != '@' )
  line++;
while ( *line )
{
  *words++ = tablePointer;
  while ( *line && (isalnum( *line ) ||
                  (index(LegalChars, *line) && isalnum( line[1] )))
    *tablePointer++ = *line++;
  *tablePointer++ = '\000';
}
if (Debug) fprintf(stderr, "IntoWords: word %s\n", *(words-1));
while ( *line && !isalnum( *line ) && *line != '/' && *line
      != '@' ) line++;
*words = NULL;

} /* IntoWords */

FoldedEQ(s1,s2,len)
register char *s1,*s2;
register len;
{
  /*
   * returns true if words match not counting case.
   * s1 can be shorter than s2, if it matches for len chars.
   */
  if (s1==NULL || s2==NULL)
    return( s1 == s2 );
  while (--len>=0 && *s1 && (isupper(*s1) ? tolower(*s1):*s1)
     == (isupper(*s2) ? tolower(*s2) : *s2))
    s1++, s2++;
  return ( len<0 ) || ( *s1==0 && *s2==0 );
} /* FoldedEQ */

Fold(s)
register char *s;
{  
  /*  
   * Convert all upper case in the string to lower  
   */  
  if (s==NULL) return;  
  while (*s) {  
    *s = isupper(*s) ? tolower(*s) : *s;  
    s++;  
  }  
  /* Fold */  
}

Execute( words )  
char **words;  
{
  /*  
   * Actually perform the indicated commands  
   */  
  if (Debug)  
  {  
    char **p = words;  
    fprintf(stderr, "Execute: Words are: ");  
    while (*p) fprintf(stderr, "%s ", *p++);  
    fprintf(stderr, "\n ");  
  }  
  /* write log message? */  
  if (logfd != NULL) {  
    char **p = words;  
    fprintf( logfd, "%s:%s:",arpadate(),User);  
    while (*p) fprintf(logfd, "%s ", *p++);  
    fprintf( logfd, "\n ");  
  }  
  #***printf("\n"); Omitted by willy. This killed HTML!****/  
  if ( FoldedEQ( *words, "please", 3) )  
    ++words;  
  if ( FoldedEQ( *words, "alias", 3) )  
    return( MakeAlias( ++words ) );  
  if ( FoldedEQ( *words, "add", 3) )  
    return( AddUser( ++words ) );  
  if ( FoldedEQ( *words, "create", 3) )
return( Create( ++words ) );

if ( FoldedEQ( *words, "delete", 3 )
    return( Delete( ++words ) );

if ( FoldedEQ( *words, "describe", 3 )
    return( Describe( ++words ) );

if ( FoldedEQ( *words, "forward", 3 )
    return( Forward( ++words ) );

if ( FoldedEQ( *words, "help", 3 )
    return( Help() );

if ( FoldedEQ( *words, "keep", 3 )
    return( Keep( ++words ) );

if ( FoldedEQ( *words, "list", 3 )
    return( List( ++words ) );

if ( FoldedEQ( *words, "maintainers", 3 )
    return( Maintainers( ++words ) );

if ( FoldedEQ( *words, "pluslots", 3 )
    return( PlusLots( ++words ) );

if ( FoldedEQ( *words, "remove", 3 )
    return( Remove( ++words ) );

if ( FoldedEQ( *words, "passwdadd", 8 )
    return( passwdadd( ++words ) );

if ( FoldedEQ( *words, "passwdchk", 8 )
    return( passwdchk( ++words ) );

if ( FoldedEQ( *words, "lpub", 3 )
    return( LPub( ++words ) );

if ( FoldedEQ( *words, "mod", 3 )
    return(Mod_list( ++words ) );

if ( FoldedEQ( *words, "all", 3 )
    return( ListAll( ++words ) );

if ( FoldedEQ( *words, "whichlist", 6 )
    return( WhichList( ++words ) );
if (FoldedEQ(*words, "whichmaint", 6))
    return(WhichMaint(++words));

printf("Invalid command: %s
", *words);
if (Debug) fprintf(stderr,"Execute: Attempting to continue\n");
    return(1);
    /* Execute */

#ifdef ALIAS_MAP
char *AliasMap = ALIAS_MAP;
char *ypDomain = NULL;

cchar *
yellowlookup(desired)
    char *desired;
{
    char *result;
    int insize, outsize;

    /*
     * if we did not find a local alias, then
     * try a remote alias through yellow pages.
     */

    if (AliasMap==NULL || *AliasMap=='\0') return(NULL);
    if (ypDomain==NULL) {
        yp_get_default_domain(&ypDomain);
        if (ypDomain == NULL) return(NULL);
    }
    insize = strlen(desired)+1;
    if (yp_match(ypDomain, AliasMap, desired, insize, &result,
                  &outsize)) {
        errno = 0;
        return(NULL);
    }

    return(result);
}
#endif

cchar *
Get(desired)
char *desired;
{

/* retrieve the given value from the main database. */
datum content, key;
static char value[MAXSIZE];
static char lowered[MAXSIZE];

key.dptr = desired;
key.dsize = strlen(desired) + 1;
key.dsize = strlen(desired); /*
content = dbm_fetch(dbm,key);
if ( content.dptr==NULL ) {
    if (Debug)
        fprintf(stderr,"Get: Value looked up for '%s'
            not found\n",desired);
strcpy(lowered,desired);
Fold(lowered);
key.dptr = lowered;
content = dbm_fetch(dbm,key);
#endif
#endif
#endif
#endif
if (strlen(content.dptr) > content.dsize)
    content.dptr[content.dsize] = '\0';
strncpy( value, content.dptr, MAXSIZE );
if (Debug) fprintf(stderr,"Get: Value looked up for %s
    is %s\n", desired, value);
return( value );
char *GetL(key, whichfile)
{
    /*
     * retrieve a line from whichfile which matches key
     * return NULL if cannot find a match
     */
    FILE *aliasFile;

    static char value[MAXSIZE];
    char line[MAXSIZE], start[MAXSIZE];
    char *filename;

    if (Debug) fprintf(stderr, "GetL: key: %s\n\twhichfile: %s\n", key, whichfile);
    /* Shitty hak */

    if (strcmp(whichfile, GROUP) == 0) {
        filename = GROUPFILE;
    } else if (strcmp(whichfile, HOME) == 0) {
        filename = HOMEFILE;
    } else if (strcmp(whichfile, EXTRA) == 0) {
        filename = EXTRAFILE;
    } else if (strcmp(whichfile, SYSTEM) == 0) {
        filename = SYSTEMFILE;
    } else if (strcmp(whichfile, LASTNAME) == 0) {
        filename = LASTNAMEFILE;
    }else if (strcmp(whichfile, LISTFILE) == 0) {
        filename = LISTFILE;
    }else if (strcmp(whichfile, "Passwd") == 0) {
        filename = PASSWDFILE;
    } else if (strcmp(whichfile, "Listname") == 0) {
        filename = (char *) malloc(128);
        strcpy(filename, ALIASDIR2);
        strcat(filename, key); /* I hope key is the list name */
    } else {
        filename = (char *) malloc(128);
        strcpy(filename, MACHINEPREFIX);
        strcat(filename, whichfile);
    }
if (Debug) fprintf(stderr,"GetL: filename:%s\n",filename);
if ((aliasFile=fopen(filename,"r")) == NULL) {
    if (Debug) fprintf(stderr,"GetL:Could not open
    the alias
    file %s for reading. Please
    report to %s.\n",filename, MAILWIZARD );

    return( NULL );
}

strcpy(start, key);
strcat(start,":");
while (fgets(line, MAXSIZE, aliasFile)) {

    if ((FoldedEQ(start,line,strlen(start)))
        ||(strcmp(whichfile,
            "Listname" )==0))
    {
        char *ind;
        if (Debug) fprintf(stderr,"GetL: line
        read:%s\n",line);
        if ((ind = index(line, '\n')) != NULL)
            *ind = '\0';
        if ((ind = index(line, ':')) == NULL)
            ind = line;
        if ((ind = index(line, ' ')) == NULL)
            /* hopefully
                this strips off leading spaces*/
            ind = line;
        else
            ind++;
        (void) strncpy(value, ind, MAXSIZE);
        if (Debug) fprintf(stderr,"GetL: returning
        %s\n",value);
        fclose(aliasFile);
        return(value);
    }
PrettyPrint(fIsList, s, sbInit)
int fIsList;
char *s, *sbInit;
{
    /*
     * Pretty print the line, starting with n characters.
     * End with a period.
     */
    int n, i;
    register char *start = s, *end = s;
    char buf[MAXSIZE];

    printf("%s", sbInit);
    for (n = strlen(sbInit); n < INITSPACE; n++)
        printf(" ");

    while (*end) {
        if (fIsList && (*end == ',')) {
            strncpy(buf, start, end - start);
            buf[end - start] = 0;
            if ((n + strlen(buf) + 2) > LINELEN) {
                printf("\n%*s", INITSPACE, " ");
                n = INITSPACE;
            }
            printf("%s", buf);
            n += strlen(buf) + 2;
            start = ++end;
            continue;
        }
    }
    if (!fIsList && (*end == ',' || (*end == '\n') || (*end == '\t')))
    {
        strncpy(buf, start, end - start);
        buf[end - start] = 0;
        if ((n + strlen(buf) + 1) > LINELEN) {
            printf("\n%*s", INITSPACE, " ");
        }
    }
}
n = INITSPACE;
}
if (*end == ' \t') {
    printf("%s",buf);
    /*skip tabs altogether */
    n += strlen(buf);
}
else {
    printf("%s ",buf);
    n += strlen(buf) + 1;
}
start = ++end;
continue;
}
*end++;
}
if (start==end)
    printf("\n");
else
    if ((n + strlen(start)) > LINELEN)
        printf("\n%*s%s\n", INITSPACE, " ", start);
    else
        printf("%s\n", start);
    } /* PrettyPrint */

ShouldBeNull( words )
char **words;
{
    /*
     * checks for extra garbage at the end of lines
     */
    if ( *words )
    {
printf( "Ignored extra garbage at the end of the line:" );
    if (Debug) fprintf(stderr, "ShouldBeNull: Ignored extra garbage at the end of the line:" );
    while ( *words ) {
        printf(" %s", *words++ );
        if (Debug) fprintf(stderr, " %s", *words++ );
    }
    printf("\n");
    return( 1 );
}
IsOn( name, list, option )
char *name, *list;
enum Option option;
{
    /*
     * Returns true if the user is on the list.
     * Note "list" is the actual list, not the @b[name] of the
     * list.
     * The option determines if the user should be deleted
     * if he is found.
     */
    register char *p=list;
    int size = strlen( name );

    while ( *p )
    {
        if ((FoldedEQ(name,p,size)) &&
            ((p[size]=='\0') || (p[size]==','))
            || (p[size]=='\n'))
        {
            /*
             * We've found the name in the list,
             * update appropriately
             */
            register char *end = p+size;

            if (option==Eliminate)
            {
                if ( p!=list ) p--;
                else if ( *end ) end++;
                while ( *end )
                {
                    *p++ = *end++;
                }
                *p = '\000';
            }
            if (Debug) fprintf(stderr, "IsOn: User %s
found on
list %s\n", name, list );
        }
        return( 1 );
    }
}
while ( *p && *p !=',' ) p++;
while ( *p==',' ) p++;
}
if (Debug) fprintf(stderr, "IsOn: Could not find user 
%s on list %s\n", name, list );
return( 0 ); /* IsOn */

Isinlist( name, list, file, option )
char *name, *list, *file;
enum Option option;
{
    /* Returns true if the user is on the list.
    * Note "list" is the actual list name, and this will
    * look for the
    * like-named file
    * The option determines if the user should be deleted
    * modified to look in whole files rather than single line
    */
    char line[MAXSIZE];
    FILE *aliasFile;
    char *filename;
    register char *p=list;
    int ison;
    int size = strlen( name );

    filename = (char*)malloc(128);
    ison=1;

    if(!strcmp(file, "Aliasdir2")){
        strcpy(filename, ALIASDIR2);
        strcat(filename, list);
    }
    else if(!strcmp(file, "Passwd")){
        /* See if this user already has a password entry */
        strcpy(filename, PASSWDFILE);
        /* printf("Checking %s file \n", filename); */
    }
    if ((aliasFile=fopen( filename, "r")) == NULL)
        printf( "Could not open the file %s for reading.\n"
while (1)
{
if( fgets( line, MAXSIZE, aliasFile )==NULL)
{
    break;
}

p=line;

if ((FoldedEQ(name,p,size)) &&
    ((p[size]=='\0') || (p[size]==','))
    || (p[size]=='\n') || (p[size]==')).))
{
    if (Debug) fprintf(stderr, "IsOn: User %s found on
    list %s\n", name, list );

    return( 1 );
}
while ( *p && *p !=',' ) p++;
while ( *p==',' ) p++;
}
if (Debug) fprintf(stderr, "IsOn: Could not find user
    %s on list %s\n", name, list );
fclose(aliasFile);
free(filename);
return( 0 );

} /* Isinlist */

IsMailingList( list, maint, description, public, date)
char *list, *maint, *description, *public, *date;
{

    /*
     * scans the mailing list file. If the name is found, also returns
     * other information about the list to the appropriate parameters.
     * Returns False if there is no such mailing list.
     */
    FILE *listFile = fopen( LISTFILE, "r" );
    char start[MAXSIZE], line[MAXSIZE], *value;
    int size;

    if (Debug) fprintf(stderr, "IsMailingList: list:%s, maint:%s,
    description:%s, public:%s, date:%s\n"
if (listFile==NULL)
{
    /*
     * bad news already. The mailing list file could
     * not be found.
     * Simulate it by looking in the alias file, and
     * assuming that
     * any alias mapping to more than one name is a
     * list.
     */
    if (Debug) fprintf(stderr,"IsMailingList: listfile not found ");
    value = GetL( list,GROUP );
    if ( value == NULL ) return( 0 );
    return( index( value, ',' ) != NULL );
}

strcpy( start, list );
strcat( start, ":" );
size = strlen(start);

while ( fgets( line, MAXSIZE, listFile ) ) {
    if ( FoldedEQ( line, start, size ) ) {
        /*
         * We've found it in the list file, now parse out
         * the information fields.
         */
        register char *begin, *end;
        int temp;

        if (Debug) fprintf( stderr, "IsMailingList
                          : %s found\n",list);
        begin = line+size;

        strcpy (maint,"");
        if ( end = index( begin, ':' ) ) {
            strncpy( maint, begin, end-begin );
            temp = end-begin;
            strcpy( maint + temp,"" );
            begin = end+1;
        } else {
            strcpy( maint, "" );
            begin = line;
        }
strcpy(public,"" );
if ( end = index( begin, '.' ) ) {
    strncpy( public, begin, end-begin );
    begin = end+1;
    strcpy( date, begin );
    if ( end=index( date, '\'n' ) )
        *end = '\000';
} else {
    strcpy( public, "" );
    strcpy( date, "" );
}

strcpy( description, "" );
while ( fgets( line, MAXSIZE, listFile )
    && line[0]=='\t')
    strcat( description, line );
fclose(listFile);
if (Debug) fprintf(stderr, "IsMailingList:"
    list: %s,
    Maint %s\n", list,
    maint );
    return( 1 );
}
fclose(listFile);
return( 0 );

/* IsMailingList */

IsMailWizard( user )
char *user;
{
    /*
     * Returns true if the user is a member of the
     * MAILWIZARD mailing list.
     */
    char *wizards = Get( MAILWIZARD );
    if ( wizards==NULL ) return( 0 );
    return( IsOn( user, wizards, Scan ) );
}

/* IsMailWizard */

GetNextLine(line)
char *line;
{  
    static int seenheaders = 0;

    while (1) {
        if (fCacheLine) {
            seenheaders = 1;
            /* templine has been leftover from
             * ReadFromLine */
            /* just use what is there */
            fCacheLine = 0;
        } else if (gets(templine)== NULL) {
            return (int)NULL ;
        }
        if ( FoldedEQ( templine, "Subject: ", 9)) {
            strcpy(line,(char *)templine+9);
            break;
        }
        if (!seenheaders && (index(templine‚::’) == 0)
            && (strlen(templine)!=0)) {
            seenheaders = 1;
            strcpy(line,templine);
            break;
        }
        if (!seenheaders && (strlen(templine)==0)) {
            seenheaders = 1;
            continue;
        }
        if (seenheaders && (strlen(templine)!=0)) {
            strcpy(line,templine);
            break;
        }
    }
}

B.2 GUI Interface Source Code

B.2.1 AppletClient.java

This is source code for the GUI file AppletClient.java

import Cool Beans.awt.PositionLayout;
import java.applet. *;
import java.lang. *;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class AppletClient extends Applet {

    public static final int PORT = 4444;
    Socket s;
    DataInputStream in;
    static PrintStream out;
    TextArea outputarea;
    StreamListener listener;
    TextField inputfield = new TextField(45);
    static String namestr = new String("Enter your email Address now!");
    static String email = new String("null");
    static String tmppstring;
    static String passwdchkEmail = new String("null");
    Stack mainstack = new Stack();
    static int isWindowShowing = 0;
    static int isWindowShowing2 = 0;
    static AddListFrame addFrame;
    static RemoveListFrame removeFrame;
    static CreateListFrame createFrame;
    static MaintainListFrame maintainFrame;
    static ModifyListFrame modifyFrame;
    static MaintainerListDeleteFrame maintainerDeleteFrame;
    static PassWordGet getPasswordFrame;
    MenuBar menu;

    //static AddListFrame addFrame = new AddListFrame
    //("Adding to a list");
    // Create a socket to communicate with a server on port
    // 4444 of the
    //host that the applet’s code is on. Create streams to use with
    // the socket. Then create a TextField for user input and a
    // TextArea
    // for server output. Finally, create a thread to wait for and
    // display server output.
    public void init() {
        setLayout(new PositionLayout());
        Panel control = new Panel();
    }
// Panel input = new Panel();

Button submail = new Button("Submit Email Address");
Button button3 = new Button("Button3");
Button add = new Button("Add to list");
Button remove = new Button("Remove from list");
Button create = new Button("Create a list");
Button maintain = new Button("Maintain a list");

// control.setLayout(new PositionLayout(250, 50));
// control.add("0 0", add);
// control.add("100 0", remove);
// control.add("0 30", create);
// control.add("100 30", maintain);
// control.setBackground(Color.gray);
// gridLayout layout = new gridLayout(2,2);
// control.setLayout(new GridLayout(2,2));
control.add(add);
control.add(remove);
control.add(create);
control.add(maintain);
add("100 100", control);
add("100 50", inputfield);

menu = new MenuBar();
// setMenuBar(menu);
Menu Help = new Menu("Help");
menu.add(Help);
MenuItem adding = new MenuItem("Adding");
MenuItem removing = new MenuItem("Removing");
MenuItem creating = new MenuItem("Creating");
MenuItem maintaining = new MenuItem("Maintaining");
Help.add(adding);
Help.add(removing);
Help.add(creating);
Help.add(maintaining);
// resize(300, 200);

try {
    // System.out.println("Beinging of Try ");
    /* this did some nifty thing to connect back to its
     * Server.. but ofcourse it didnt work, I hard
     * coded it in */
    // s = new Socket("jarok.cs.ohiou.edu", PORT);
    s = new Socket(this.getCodeBase().getHost(), PORT);
in = new DataInputStream(s.getInputStream());
out = new PrintStream(s.getOutputStream());
outputarea = new TextArea();
outputarea.setEditable(false);

add("0 225", outputarea);
listener = new StreamListener(in, outputarea);

this.showStatus("Connected to "+s.getInetAddress().getHostAddress()
+ ":" + s.getPort());
//System.out.println("End of Try");
} catch (IOException e) { this.showStatus(e.toString()); }
}

public void paint(Graphics g){
    g.drawString(namestr,30,10);
}

static public void StackKill(){
    while(StreamListener.listStack.empty()==false)
        StreamListener.listStack.pop();
}

static public void test(){
    //System.out.println("IN DA TEST! \n");
    if(StreamListener.listStack.search(":Remove from:")!= -1){
        removeFrame = new RemoveListFrame("Removing from a list");
        removeFrame.show();
    }
    else if(StreamListener.listStack.search(":Add to:")!= -1){
        addFrame = new AddListFrame("Adding to a list");
        addFrame.show();
    }
    else if(StreamListener.listStack.search(":ListsMaintained:"))
        !=-1){
        maintainFrame = new MaintainListFrame("Pick one of your
else if(StreamListener.listStack.search(":false password:") !=-1){
    namestr="Invalid Password for "+passwdchkEmail;
    email="null";
    AppletClient.StackKill();
    System.out.println("Password just got rejected");
}
else if(StreamListener.listStack.search(":true password:")
    !=-1){
    email=passwdchkEmail;
    namestr="OU Mailer for: "+email;
    AppletClient.StackKill();
    System.out.println("Password got accepted ");
namestr="OU Mailer for: "+inputfield.getText();
email=inputfield.getText();
repaint();

if(what=="Add to list"){
    if(email.equals("null") == false){ // if not equal to null
        if(isWindowShowing == 0){
            //
            System.out.println("Lpub");
            // isWindowShowing = 1;
            out.println("Lpub"); /* lists all mailing lists*/
            /* and tells if their public or not*/
            return true;
        }
    }
    return true;
}

if(what=="Remove from list"){
    if(email.equals("null") == false){ // if not equal to null
        if(isWindowShowing == 0){
            // System.out.println("Removeing from");
            //isWindowShowing = 1;
            out.println("whichlist "+email);
            return true;
        }
    }
    return true;
}

if(what == "Create a list"){
    if(email.equals("null") == false){ // if not equal to null
        if(isWindowShowing == 0){
            // isWindowShowing = 1;
            createFrame = new CreateListFrame("Creating a list");
            createFrame.show();
        }
    }
    return true;
}

if(what == "Maintain a list"){
    if(isWindowShowing == 0 && email.equals("null") == false){
/**
 * Bring up checkpasswd box */
tmpstring=inputfield.getText();
if(isWindowShowing == 0 && tmpstring.equals("null")
== false){
    getPasswordFrame = new PassWordGet("Enter Password"
     ,tmpstring);
    getPasswordFrame.show();
    return true;
}
return false;
}

class PassWordGet extends Frame{
    TextField password = new TextField(10);
    Panel control = new Panel();
    String useremail,pass;
    String tmp,fullemail;

    PassWordGet(String title, String email){
        super(title);
        AppletClient.isWindowShowing = 1;
        fullemail = email;{/* Needed to have the full email addy in the
            action section*/
        useremail = nameParse(email);
        GridLayout layout = new GridLayout(3,1);
        setLayout (layout);
        password.setEchoCharacter('*');
        add(new Label("Enter Password for "+useremail,Label.CENTER));
        add(password);
        Button cancel = new Button("Cancel");
        Button ok = new Button("Ok");
        control.add(cancel);
        control.add(ok);
        add(control);
public String nameParse(String name) {
    int i;

    for (i = 0; i < name.length(); i++) {
        if (name.charAt(i) == '62')
            return (name.substring(0, i));
    }
    return (name);
}

public boolean action(Event event, Object what) {
    if (what == "Cancel") {
        AppletClient.isWindowShowing = 0;
        dispose();
    }
    if (what == "Ok" || event.target == password) {
        pass = password.getText();
        AppletClient.passwdchkEmail = fullemail;
        AppletClient.out.println("passwdchk " + useremail + " " + pass);
        AppletClient.isWindowShowing = 0;
        dispose();
    }
    return true;
}

class AddListFrame extends Frame{
    static Panel addbox = new Panel();
    static Panel ok = new Panel();
    static int addboxcount;
    //static Checkbox[] cb = new Checkbox[20];
    static Vector cb = new Vector();
    Button button;
    MenuBar menu;

    AddListFrame(String title){

super(title);
String buttonName = new String();
AppletClient.isWindowShowing = 1;
addboxcount = 0;
while("Add to:".equals((String)
StreamListener.listStack.peek())==false){

buttonName = (String)
StreamListener.listStack.pop();
if(buttonName.equals("End:"))
}
else{
//System.out.println(buttonName);
if(buttonName.startsWith("priv:"))
else{
    cb.insertElementAt(new Checkbox
(buttonName.substring(5)),addboxcount);
    addbox.add((Checkbox)cb.elementAt
(addboxcount++));
}
}
StreamListener.listStack.pop();
    // pop off that :Add to:
Button addSubmit =
    new Button("Add me to these lists");
Button close = new Button("Close");
ok.add(close);
ok.add(addSubmit);

		/* This is where I do my formatting.*/
		/* I found that is I dont specifically format
		* my window, it comes up blank */
GridLayout layout =
    new GridLayout((addboxcount/3)+1,3);
addbox.setLayout(layout);
BorderLayout layout2 = new BorderLayout();
setLayout(layout2);
add("North",new Label("Pick the lists you wish
    to join."));
add("Center",addbox);
add("South",ok);

		//*** Menu Stuff ***/
menu = new MenuBar();
setMenuBar(menu);
Menu Help = new Menu("Help");
MenuItem addhelp =
    new MenuItem("Adding yourself to a list");
Help.add(addhelp);
menu.add(Help);
menu.setHelpMenu(Help);
pack();

public boolean action(Event event, Object what){
    int i;

    if(what == "Add me to these lists"){
        for(i=0; i < addboxcount;i++){
            if(((Checkbox) cb.elementAt(i)).getState()){
                AppletClient.out.println("add "+AppletClient.email+" to 
" +(Cb.box.elementAt(i)).getLabel()+" with " 
+AppletClient.email);
            }
            else{
            }
        }
        AppletClient.isWindowShowing = 0;
dispose();
addbox.removeAll();
ok.removeAll();
    }

    if(what == "Adding yourself to a list"){
        InfoDialog d = new InfoDialog(AppletClient.addFrame,
            "Help for adding yourself to a list",
            "This will have adding help");
    }
    if(what == "Close"){
        addbox.removeAll();
        ok.removeAll();
        AppletClient.isWindowShowing = 0;
dispose();
    }
AppletClient.StackKill();
    return true;
}

// Wait for output from the server on the specified stream,
// and display
// it in the specified TextArea.

class StreamListener extends Thread {
    DataInputStream in;
    TextArea output = new TextArea();
    static Stack listStack = new Stack();

    public StreamListener(DataInputStream in, TextArea output) {
        this.in = in;
        this.output = output;
        this.start();
    }
    public void run() {
        String line = new String();

        //System.out.println("In Run");

        try {
            for(;;) {
                line = in.readLine();
                if (line == null) break;
                //output.setText(line);
                if(!line.equals("null")
                   && !line.equals("Unknown command:"))
                    listStack.push(line);
                if(line.equals(":End:"))
                    AppletClient.test();

                if(line.equals("The user " + AppletClient.email + " is a
                    maintainer of the following lists:")){
                    listStack.push("Begin WhichMaint:");
                    for(line = in.readLine();
                        line.equals(":End of WhichMaint:")
                        == false; line = in.readLine()){
                        listStack.push(line);
                    }
                    listStack.push("ListsMaintained:");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
AppletClient.test();
}
else if(line.equals("The user "+AppletClient.email
" is not a maintainer of any lists.")){
    System.out.println(" User dosent own any lists");
    AppletClient.isWindowShowing = 0;
}
if(line.equals(":Maintainer Delete:"))
    AppletClient.test();
if(line.equals("Members ::Maintainer Delete:"))
    AppletClient.test();
if(line.equals(":true password:"))
    AppletClient.test();
if(line.equals(":false password:"))
    AppletClient.test();
if(line.equals("You have tried to create a list
    that already exists!Please try another name.")){
    CreateListFrame.modifyFrame.dispose();
    AppletClient.isWindowShowing = 0;
}
output.appendText(line+"\n");
}
}
catch (IOException e) { output.setText(e.toString()); }
finally {
    output.appendText("Connection closed by server.");
}
}

class RemoveListFrame extends Frame{
    static Panel removebox = new Panel();
    static Panel ok = new Panel();
    static int removeboxcount;
    //static Checkbox[] cb = new Checkbox[20];
    static Vector cb = new Vector();
    Button button;
    MenuBar menu;

    RemoveListFrame(String title){
        super(title);
    }
String buttonName = new String();
AppletClient.isWindowShowing = 1;
//System.out.println("I am in RemoveListFrame..."瘙
removeboxcount=0;
while("":Remove from:":equals
((String)StreamListener.listStack.peek())==false){

    buttonName = (String)
            StreamListener.listStack.pop();
    if(buttonName.equals("":End:"") == true){
        //System.out.println("End found");
    }
    else{
        cb.insertElementAt( new Checkbox(buttonName)
 , removeboxcount);
        removebox.add((Checkbox)cb
 .elementAt(removeboxcount++));
    }
}
StreamListener.listStack.pop();
// pop off that :Remove from:
Button Submit =
    new Button("Remove me from these lists");
Button close = new Button("Close");
ok.add(close);
ok.add(Submit);
// removebox.setBackground(Color.green);
/*** This is Help Menu stuff****/
menu = new MenuBar();
setMenuBar(menu);
Menu Help = new Menu("Help");
MenuItem delhelp =
    new MenuItem("Deleting your self from a list");
Help.add(delhelp);
menu.add(Help);
menu.setHelpMenu(Help);
/*** End help menu stuff ***/

/* This is where I do my formatting.*/
/* I found that is I dont specifically format
my window,it comes up blank */
GridLayout layout =
    new GridLayout((removeboxcount/3)+1,3);
removebox.setLayout(layout);
add("0 0",new Label("Pick the lists you wish

*/
to be removed from")
BorderLayout layout2 = new BorderLayout();
setLayout(layout2);
add("North",new Label("Check off the lists you
want removed from"));
add("Center",removebox);
add("South",ok);
pack();

}

public boolean action(Event event, Object what){
    int i;

    if(what == "Remove me from these lists"){
        for(i=0; i < removeboxcount;i++){
            if(((Checkbox)cb.elementAt(i)).getState()){
                AppletClient.out.println("delete "+
                +AppletClient.email+" from "+
                +((Checkbox)cb.elementAt(i)).getLabel()+" with "+
                +AppletClient.email);
            }
        }
    }else{
    }
    AppletClient.isWindowShowing = 0;
dispose();
removebox.removeAll();
ok.removeAll();
}

if(what == "Deleting your self from a list"){
    InfoDialog d = new InfoDialog(AppletClient.removeFrame,
    "Help for Deleting your self from a list",
    "This is a list of all the mailing lists 
"+"you are on. Check off the lists you want to 
"+"remove yourself from, then hit the 'Remove 
"+"myself from these lists' button. 
"");
    d.show();
}

if(what == "Close"){
    removebox.removeAll();
    ok.removeAll();
AppletClient.isWindowShowing = 0;
dispose();
}
AppletClient.StackKill();
return true;
}

/***
Create list ***/
class CreateListFrame extends Frame{
    static Panel okpanel = new Panel();
    //static int removeboxcount;
    TextField inputfield = new TextField(30);
    Button button;
    static ModifyListFrame modifyFrame;

    CreateListFrame(String title){
        super(title);
        String buttonName = new String();
        AppletClient.isWindowShowing = 1;
        //System.out.println("I am in CreateList");
        // removeboxcount=0;

        /* This is where I do my formatting.*/
        /* I found that is I dont specifically format
         * my window, it comes up blank */
        BorderLayout layout2 = new BorderLayout();
        setLayout(layout2);
        Button close = new Button("Close");
        Button ok = new Button("Ok");
        add("North",new Label("Enter the name of the list "
            +AppletClient.email+ " wishes to maintain.");
        okpanel.add(close);
        okpanel.add(ok);
        add("South",okpanel);
        add("Center", inputfield);
        pack();
    }
}

public boolean action(Event event, Object what){
    int i;
    String list;
if (event.target == inputfield || what == "Ok") {
    list = inputfield.getText();
    modifyFrame = new ModifyListFrame("Modifying a list", list, 1);
    modifyFrame.show();
    AppletClient.out.println("create " + list + " with "+ AppletClient.email);
    okpanel.removeAll();
    dispose();
}
if (what == "Close") {
    AppletClient.isWindowShowing = 0;
    okpanel.removeAll();
    dispose();
}
return true;
}

/***
Get Moderated password **/
/* I dont really like this function
I couldnt have it JUST get a passwd
because the parent procedure wouldnt
wait for the user to give one. When I
had my parent call to see if this frame
was still showing, I ran into the classic
polling problem. I could not find a
usefull sleep function to sue. Finally
I decided to pass more arguments to this procedure,
and have IT call to the java server */
class GetModeratedPasswd extends Frame{
    static String passwd = new String();
    public String passwdck = new String();
    public TextField passwdfield = new TextField(10);
    public TextField passwdcheck = new TextField(10);
    Panel control = new Panel();
    public int Ion;
    public String Listname;
    GetModeratedPasswd(String title, int ion, String listname){
    /* 0 is ioff, 1 is ion **/
        super(title);
        Ion = ion;
        Listname = listname;
        GridLayout layout = new GridLayout(6, 1);
        setLayout(layout);
        passwdfield.setEchoCharacter('*');
passwdcheck.setEchoCharacter('*');
add(new Label("Enter a Password to be used", Label.CENTER));
add(passwdfield);
add(new Label("Re-enter the password", Label.CENTER));
add(passwdcheck);

Button close = new Button("Close");
Button clear = new Button("Clear");
Button ok = new Button("Ok");
control.add(close);
control.add(clear);
control.add(ok);
add(control);
pack();
}

public boolean action(Event event, Object what){

if (what == "Close"){
    dispose();
    control.removeAll();
    AppletClient.isWindowShowing2 = 0;
    return true;
}

if (what == "Clear"){
    passwdfield.setText("" Cheers;
    passwdcheck.setText(""");
    return true;
}

if (what == "Ok" || event.target == passwdcheck){
    passwd = passwdfield.getText();
    passwdck = passwdcheck.getText();
    if (passwd.equals(passwdck) == false){
        //System.out.println("Passwords do not match");
        add(new Label("Passwords do not match"));
    }
    else{
        ModifyListFrame.passwd = passwd;
        if(Ion==0)
            /* Ion is off, list won't be mailable only by members */
            AppletClient.out.println("mod " + Listname+" with "
                          + AppletClient.email+" ioff aon "+ passwd);
        if(Ion==1)
/*Ion is on, list will only be mailable by members */
AppletClient.out.println("mod "+Listname+" with "
+AppletClient.email+" ion aon "+ passwd);
dispose();
AppletClient.isWindowShowing2 = 0;
}
return true;
}
return true;
}

/*** Modify list ***/
class ModifyListFrame extends Frame{
    static Panel removebox = new Panel();
    static int removeboxcount;
    Button button;
    static TextField addField = new TextField(45);
    Button removeButton = new Button("Remove User");
    Button closeButton = new Button("Close");
    Button delList = new Button("Delete this list");
    Button makeButton = new Button("Make Modifications");
    Button addUserButton = new Button("Add User:");
    static String addcoment =new String("Waiting for user to add");
    CheckboxGroup ls = new CheckboxGroup();
    public Checkbox priv = new Checkbox("Private",ls,false);
    public Checkbox pub = new Checkbox("Public",ls,false);
    public Checkbox moderated = new Checkbox("Moderated");
    public Checkbox mailbymem =
        new Checkbox("Only mailable by members");
    static String listname;
    static String passwd;
    static GetModeratedPasswd passFrame;
    static int Parent;
    MenuBar menu;
    ModifyListFrame(String title, String list, int parent){
        super(title);
        Parent = parent;
        String buttonName = new String();
        //System.out.println("I am in ModifyListFrame...");

        listname= list;
        add("10 10",new Label("Modifying Da list");

        /* This is where I do my formatting.*/
    }
/* I found that is I dont specifically format my window, it comes up blank */
setLayout(new PositionLayout());

/* making list state panel */

Panel liststate = new Panel();
GridLayout layout =
    new GridLayout(removeboxcount, 1);
    liststate.setLayout(layout);

    liststate.add(priv);
    liststate.add(pub);

/* Done with state panel */
/* making the modifications panel */

Panel modif = new Panel();
modif.setLayout(layout);

    modif.add(mailbymem);
    modif.add(moderated);

/**********************/
resize(600, 600);
add("50 10", new Label("Modifying list: " + list));
add("50 50", modif);
add("150 100", addField);
add("50 100", addUserButton);
add("250 200", removeButton);
add("50 200",
    new Label("Press here to remove a user: "));
add("270 50", liststate);
add("150 250", closeButton);
add("225 250", delList);
add("350 50", makeButton);
menu = new MenuBar();
setMenuBar(menu);

Menu Help = new Menu("Help");

//menu.add(Help);

MenuItem mods =
    new MenuItem("Making Modifications");
MenuItem adding = new MenuItem("Adding a user");
MenuItem removing =
    new MenuItem("Removing a user");
MenuItem deleting =
    new MenuItem("Deleting the list");
Help.add(mods);
Help.add(adding);
Help.add(removing);
Help.add(deleting);
menu.add(Help);
menu.setHelpMenu(Help);
pack();

//button = new Button("Close Window");
//add(button);

public void paint(Graphics g){
g.drawString("Add Status: "+addComment,75,240);
/*this is the add comment */
}

public boolean action(Event event, Object what){
int i;
Checkbox tmp;
String user;

if(what=="Close"){
    AppletClient.isWindowShowing = 0;
dispose();
    // return true;
}

if(what == "Delete this list"){
    AppletClient.isWindowShowing = 0;
    AppletClient.out.println("delete "+listname+" with "
    +AppletClient.email);
dispose();
    // return true;
}

if(what=="Remove User"){
    //System.out.println("Remove user pressed");
    if(AppletClient.isWindowShowing2 == 0){
        // AppletClient.isWindowShowing2 = 1;
        // moved to maintainerdelete
        AppletClient.out.println("list "+listname);
//return true;
}
else
    System.out.println("Remove user isnt working cause
iswindowshowing = "+AppletClient.isWindowShowing2);
}

if(what=="Make Modifications"){
    //System.out.println("Make mods pressed");
    if(moderated.getState()==true
        && mailbymem.getState()==true){
        if(AppletClient.isWindowShowing2 == 0){
            AppletClient.isWindowShowing2 = 1;
            passFrame =
                new GetModeratedPasswd("Making passwd for
                "+listname,1,listname);
            passFrame.show();
        }
    }
    }
else if(moderated.getState()== true
        && mailbymem.getState()==false){
        if(AppletClient.isWindowShowing2 == 0){
            AppletClient.isWindowShowing2 = 1;
            passFrame =
                new GetModeratedPasswd("Making passwd for
                "+listname,0,listname);
            passFrame.show();
        }
    }
else if(moderated.getState()== false
        && mailbymem.getState()==true){
        AppletClient.out.println("mod "+listname+" with "
            +AppletClient.email+" ion aoff");
        // Do mailbymem only
    }
else if(moderated.getState()== false
        && mailbymem.getState()==false){
        AppletClient.out.println("mod "+listname+" with "
            +AppletClient.email+" ioff aoff");
        // shut all off
    }
if (ls.getCurrent() != null) {
    tmp = ls.getCurrent();
    if (tmp.getLabel() == "Public")
        AppletClient.out.println("add public to "+listname
+ " with "+AppletClient.email);
    else if (tmp.getLabel() == "Private")
        AppletClient.out.println("delete public from "+listname
+ " with "+AppletClient.email);
    else
        System.out.println("Problem with pub/priv got: "
+ tmp.getLabel());
}

    // return true;
}

if (what == "Add User:" || event.target == addField) {
    user = addField.getText();
    if (user.equals("") |
        addComment = "Must specify a email address to add";
    repaint();
    //return true;
}

AppletClient.out.println("add "+user+" to "+listname
+ " with "+AppletClient.email);
addComment = "Added "+user+" to mailinglist "+listname;
addField.setText(""); //Clear out the add field
repaint();
System.out.println("Add user button pressed");
}

**** Help stuff ***/

if (what == "Making Modifications"){
    if (Parent == 1) {
        InfoDialog d =
            new InfoDialog(CreateListFrame.modifyFrame, "Help for Making Modifications",
            "You can make a list mailible only by the people on the list\n"+ "by checking the 'mailable by members' box. You can make all\n"+ "submisions to a list come first to the owner for approval before \n"+ "the rest of the list recieves the submission by checking the\n"+ "Moderated box. Note: this requires you enter a password to \n"+ "approve the submissions with. To approve the submission add the\n"+ "line Approved: [password] to the first line of the e-mail, and\n"+"}
"forward that e-mail back to the mailing list. \n"
"You may also make a list joinable to the public, or make it so\n"only the owner may add users, by using the Public/Private radio
\n"button. Lists start out being private, and making no choice
will\n"leave the list as it was.");

d.show();
}
else if (Parent ==2 ){
    InfoDialog d=
        new InfoDialog(MaintainListFrame.modifyFrame,
"Help for Making Modifications",
"You can make a list mailible only by the people on the list\n" +
"by checking the 'mailable by members' box. You can make all\n" +
"submissions to a list come first to the owner for approval before
\n"the rest of the list recieves the submission by checking the\n" +
"Moderated box. Note: this requires you enter a password to \n" +
"approve the submissions with. To approve the submission add the\n" +
"line Approved: [password] to the first line of the e-mail, and\n" +
"forward that e-mail back to the mailing list. \n" +
"You may also make a list joinable to the public, or make it so\n" +
"only the owner may add users, by using the Public/Private
radio\n" +
"button. Lists start out being private, and making no choice
will\n"leave the list as it was.");

d.show();
}
//return true;
}
if(what == "Adding a user"){

    // Hard wirering these parents into this frame.
    // I couldnt seem to do it eliquently
    //This frame can have 2 different frames as its parent.
    if(Parent ==1 ){
        InfoDialog d =
            new InfoDialog(CreateListFrame.modifyFrame ,
"Help for Adding Users",
"You may add users one at a time\n" +
"by entering a user name into the add.\n" +
"input box, and clicking the Add user button.");
    d.show();
}
else if (Parent == 2 ) { 
    InfoDialog d =
    new InfoDialog(MaintainListFrame.modifyFrame,
    "Help for Adding Users",
    "You may add users one at a time\n" +
    "by entering a user name into the add.\n" +
    "input box, and clicking the Add user button.");
    d.show();
} 
    // return true;

if (what == "Removing a user") { 
    if (Parent == 1 ) { 
        InfoDialog d =
        new InfoDialog(CreateListFrame.modifyFrame,
        "Help for Removing Users",
        "You may remove users one at a time\n" +
        "by clicking the 'Remove User' button,\n" +
        "choosing the people you want to remove,\n" +
        "then pressing the 'Remove these people\n" +
        "from the list' button");
        d.show();
    } 
    else if (Parent == 2 ) { 
        InfoDialog d =
        new InfoDialog(MaintainListFrame.modifyFrame,
        "Help for Removing Users",
        "You may remove users one at a time\n" +
        "by clicking the 'Remove User' button,\n" +
        "choosing the people you want to remove,\n" +
        "then pressing the 'Remove these people\n" +
        "from the list' button");
        d.show();
    } 
    // return true;
}
if (what == "Deleting the list") { 
    if (Parent == 1 ) { 
        InfoDialog d =
        new InfoDialog(CreateListFrame.modifyFrame,
        "Help for Deleting the list",
        "You may delete the list one at a time\n" +
        "by clicking the 'Delete User' button,\n" +
        "choosing the people you want to delete,\n" +
        "then pressing the 'Delete these people\n" +
        "from the list' button");
        d.show();
    } 
    else if (Parent == 2 ) { 
        InfoDialog d =
        new InfoDialog(MaintainListFrame.modifyFrame,
        "Help for Deleting the list",
        "You may delete the list one at a time\n" +
        "by clicking the 'Delete User' button,\n" +
        "choosing the people you want to delete,\n" +
        "then pressing the 'Delete these people\n" +
        "from the list' button");
        d.show();
    } 
    // return true;
}
"By clicking the 'Delete this list' button
You will delete this mailing list from the
data base.");
d.show();
}
else if (Parent ==2 ){
    InfoDialog d =
        new InfoDialog(MaintainListFrame.modifyFrame,
        "Help for Deleting the list",
        "By clicking the 'Delete this list' button
        You will delete this mailing list from the
        data base.");
    d.show();
}
//return true;
}

//System.out.println("In Modify action");
AppletClient.StackKill();
return true;
}

class MaintainListFrame extends Frame{
    static Panel listBox = new Panel();
    static Panel buttonPanel = new Panel();
    static int listBoxCount;
    //static Checkbox[] cb = new Checkbox[20];
    static Vector cb = new Vector();
    public CheckboxGroup boxgroup = new CheckboxGroup();
    Button ok;
    Button close;
    static ModifyListFrame modifyFrame;
    String listString = new String();
    StringBuffer tmplist = new StringBuffer();

    MaintainListFrame(String title){
        super(title);
        String buttonName = new String();
        AppletClient.isWindowShowing = 1;
        //System.out.println("I am in MaintainListFrame");
        listBoxcount=0;
    }
/* This is where I do my formatting. */
/* I found that is I dont specifically format
   my window, it comes up blank */
if(".ListsMaintained:".equals((String)
    StreamListener.listStack.pop())==true){
    for(listString=((String)StreamListener
        .listStack.pop());listString
        .equals(".Begin WhichMaint:")==false;
        listString=(String)StreamListener
        .listStack.pop()){
        int i;

        for(i=0;i<listString.length();i++){
            if(\',\' == (listString.charAt(i))==false) {
                if(\',\' == (listString.charAt(i))==false)
                    tmplist.append(listString.charAt(i));
            } else{
                cb.insertElementAt(new Checkbox(tmplist
                    .toString(),boxgroup,false), listboxcount);
                listbox.add((Checkbox)cb
                    .elementAt(listboxcount++));
                tmplist.setLength(0);
            }
        }
    }
} /*should be another button to go*/

ok = new Button("Ok");
close = new Button("Close");
buttonPanel.add(close);
buttonPanel.add(ok);

GridLayout layout =
    new GridLayout(((listboxcount/3)+1)*25);
//System.out.println("listboxcount ="+listboxcount);
listbox.setLayout(layout);
GridLayout layout2 = new GridLayout(1,2);
buttonPanel.setLayout(layout2);
//GridLayout layout3 = new GridLayout(2,1);
setLayout(new PositionLayout());
//setLayout(layout3);
int i = (((listboxcount/3)+1)*25);
public boolean action(Event event, Object what){
    int i;
    String list;
    Checkbox tmp;

    if (what == "Ok"){
        if (boxgroup.getCurrent() == null) {
            listbox.removeAll();
            buttonPanel.removeAll();
            AppletClient.isWindowShowing = 0;
            dispose();
            // return true;
        }
        else{
            tmp = boxgroup.getCurrent();
            modifyFrame = new ModifyListFrame("Modifying " + tmp.getLabel(),
                                               tmp.getLabel(), 2);
            modifyFrame.show();
            listbox.removeAll();
            buttonPanel.removeAll();
            dispose();
            // return true;
        }
    }
    else if (what == "Close") {
        listbox.removeAll();
        buttonPanel.removeAll();
        AppletClient.isWindowShowing = 0;
        dispose();
        // return true;
    }
    AppletClient.StackKill();
    return true;
}

//public class StackKill extends Applet{
//
// StackKill(){
while(StreamListener.listStack.empty()==false)
// System.out.println("Killing "+StreamListener.listStack.pop());
//}
//}}
class MaintainerListDeleteFrame extends Frame{
    static Panel removebox = new Panel();
    static Panel ok = new Panel();
    static int removeboxcount;
    //static Checkbox[] cb = new Checkbox[20];
    static Vector cb = new Vector();
    Button button;

    MaintainerListDeleteFrame(String title)
    {
        super(title);
        AppletClient.isWindowShowing2 = 1;
        /* I moved isWindowShowing2 up here because it IS possible to
         * hit the remove users button, but have some other function kill
         * the stack before my listener and return vaule tester see's the
         * vaules that set this window off. By moveing the value
         * to this class,
         * atleast if some problem pops up, then can just re hit
         * the button.
         * Before, rehitting the button wouldnt work
         * because isWindowShowing was =1 */
        String buttonName = new String();
        String tmp = new String();

        removeboxcount=0;
        // The first thing on the stack should be :Maintainer Delete:
        // lets strip it off

        if((StreamListener.listStack.peek().toString().
            startsWith("Members ::Maintainer Delete:"))
            == false)
        StreamListener.listStack.pop();
        while((StreamListener.listStack.peek().
            toString().startsWith("Members :")) == false){
            buttonName =
                (String)StreamListener.listStack.pop();
            cb.insertElementAt(
                new Checkbox(buttonName),removeboxcount);
removebox.add(((Checkbox)cb.elementAt
(removeboxcount++));
}
buttonName = StreamListener.listStack.pop()
.toString().substring(9);
// peel off that Members :
cb.addElementAt(new Checkbox(buttonName)
, removeboxcount);
removebox.add(((Checkbox)cb
.elementAt(removeboxcount));
removeboxcount++;
Button removeSubmit =
new Button("Remove these people from the list");
ok.add(removeSubmit);
} /*End of emptylist if */
Button close = new Button("Close");
ok.add(close);

BorderLayout layout2 = new BorderLayout();
setLayout(layout2);
add("North", new Label("Pick the users you wish
to delete from " + ModifyListFrame.listname));
/* This is where I do my formatting. */
/* I found that is I dont specifically format
my window, it comes up blank */

GridLayout layout =
new GridLayout((removeboxcount/2), 2);
removebox.setLayout(layout);

add("Center", removebox);
add("South", ok);
pack();

}

public boolean action(Event event, Object what){
    int i;

    if (what == "Close"){
        removebox.removeAll();
    }
ok.removeAll();
AppletClient.isWindowShowing2 = 0;
dispose();
}
if (what == "Remove these people from the list") {
    //System.out.println("In maintainers delete from list ");
    for (i = 0; i < removeboxcount; i++) {
        if (((Checkbox) cb.elementAt(i)).getState()) {
           AppletClient.out.println("delete "+((Checkbox) cb.elementAt(i)).
                           .getLabel()+" from "+ModifyListFrame.listname
                           +" with "+AppletClient.email);
        }
    }
else {
    }
}
dispose();
AppletClient.isWindowShowing2 = 0;
removebox.removeAll();
ok.removeAll();
}
AppletClient.StackKill();
return true;
}

B.2.2 InfoDialog.java

This is source code for the GUI file InfoDialog.java

// This example is from the book _Java in a Nutshell_
// by David Flanagan.
// Written by David Flanagan. Copyright (c) 1996 O'Reilly &
// Associates.
// You may study, use, modify, and distribute this example
// for any purpose.
// This example is provided WITHOUT WARRANTY either expressed
// or implied.

import java.awt.*;
public class InfoDialog extends Dialog {
    protected Button button;
    protected MultiLineLabel label;

    public InfoDialog(Frame parent, String title, String message) {
        // Create a dialog with the specified title
        super(parent, title, false);

        // Create and use a BorderLayout manager with specified margins
        this.setLayout(new BorderLayout(15, 15));

        // Create the message component and add it to the window
        label = new MultiLineLabel(message, 20, 20);
        this.add("Center", label);

        // Create an Okay button in a Panel; add the Panel to the window
        // Use a FlowLayout to center the button and give it margins.
        button = new Button("Ok");
        Panel p = new Panel();
        p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
        p.add(button);
        this.add("South", p);

        // Resize the window to the preferred size of its components
        this.pack();
    }

    // Pop down the window when the button is clicked.
    public boolean action(Event e, Object arg) {
        if (e.target == button) {
            this.hide();
            this.dispose();
            return true;
        }
        else return false;
    }

    // When the window gets the keyboard focus, give it to the button.
    // This allows keyboard shortcuts to pop down the dialog.
    public boolean gotFocus(Event e, Object arg) {
        button.requestFocus();
        return true;
    }
}

B.2.3 MultiLineLabel.java

This is source code for the GUI file MultiLineLabel.java

// This example is from the book _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan. Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY either expressed or implied.

import java.awt.*;
import java.util.*;

public class MultiLineLabel extends Canvas {
  public static final int LEFT = 0; // Alignment constants
  public static final int CENTER = 1;
  public static final int RIGHT = 2;
  protected String[] lines;  // The lines of text to display
  protected int num_lines;    // The number of lines
  protected int margin_width; // Left and right margins
  protected int margin_height; // Top and bottom margins
  protected int line_height;  // Total height of the font
  protected int line_ascent;  // Font height above baseline
  protected int[] line_widths; // How wide each line is
  protected int max_width;    // The width of the widest line
  protected int alignment = LEFT; // The alignment of the text.

  // This method breaks a specified label up into an array of lines.
  // It uses the StringTokenizer utility class.
  protected void newLabel(String label) {
    StringTokenizer t = new StringTokenizer(label, "\n");
    num_lines = t.countTokens();
    lines = new String[num_lines];
    line_widths = new int[num_lines];
    for(int i = 0; i < num_lines; i++) lines[i] = t.nextToken();
  }

  // This method figures out how the font is, and how wide each line of the label is, and how wide the widest line is.
protected void measure() {
    FontMetrics fm = this.getFontMetrics(this.getFont());
    // If we don't have font metrics yet, just return.
    if (fm == null) return;

    line_height = fm.getHeight();
    line_ascent = fm.getAscent();
    max_width = 0;
    for(int i = 0; i < num_lines; i++) {
        line_widths[i] = fm.stringWidth(lines[i]);
        if (line_widths[i] > max_width) max_width = line_widths[i];
    }
}

// Here are four versions of the constructor.
// Break the label up into separate lines, and save
// the other info.
public MultiLineLabel(String label, int margin_width,
                      int margin_height,
                      int alignment) {
    newLabel(label);
    this.margin_width = margin_width;
    this.margin_height = margin_height;
    this.alignment = alignment;
}
public MultiLineLabel(String label, int margin_width,
                      int margin_height) {
    this(label, margin_width, margin_height, LEFT);
}
public MultiLineLabel(String label, int alignment) {
    this(label, 10, 10, alignment);
}
public MultiLineLabel(String label) {
    this(label, 10, 10, LEFT);
}

// Methods to set the various attributes of the component
public void setLabel(String label) {
    newLabel(label);
    measure();
    repaint();
}
public void setFont(Font f) {
    super.setFont(f);
    measure();
    repaint();
public void setForeground(Color c) {
    super.setForeground(c);
    repaint();
}

public void setAlignment(int a) {
    alignment = a;
    repaint();
}

public void setMarginWidth(int mw) {
    margin_width = mw;
    repaint();
}

public void setMarginHeight(int mh) {
    margin_height = mh;
    repaint();
}

public int getAlignment() {
    return alignment;
}

public int getMarginWidth() {
    return margin_width;
}

public int getMarginHeight() {
    return margin_height;
}

// This method is invoked after our Canvas is first created
// but before it can actually be displayed. After we’ve
// invoked our superclass’s addNotify() method, we have font
// metrics and can successfully call measure() to figure out
// how big the label is.
public void addNotify() {
    super.addNotify();
    measure();
}

// This method is called by a layout manager when it wants to
// know how big we’d like to be.
public Dimension preferredSize() {
    return new Dimension(max_width + 2*margin_width,
                         num_lines * line_height + 2*margin_height);
}

// This method is called when the layout manager wants to know
// the bare minimum amount of space we need to get by.
public Dimension minimumSize() {
    return new Dimension(max_width, num_lines * line_height);
}

// This method draws the label (applets use the same method).
// Note that it handles the margins and the alignment, but that
// it doesn’t have to worry about the color or font—the
// superclass
// takes care of setting those in the Graphics object
// we’re passed.
public void paint(Graphics g) {
    int x, y;
    Dimension d = this.size();
    y = line_ascent + (d.height - num_lines * line_height)/2;
    for(int i = 0; i < num_lines; i++, y += line_height) {
        switch(alignment) {

case LEFT:  
    x = margin_width; break;

case CENTER:
    default:
        x = (d.width - line_widths[i])/2; break;

case RIGHT:
    x = d.width - margin_width - line_widths[i]; break;

}  
g.drawString(lines[i], x, y);
}
}

B.3 GUI Server Source Code

B.3.1 KKMultiServer.java

This is source code for the GUI Server file KKMultiServer.java

import java.net.*;
import java.io.*;

class KKMultiServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        boolean listening = true;

        try {
            serverSocket = new ServerSocket(4444);
        } catch (IOException e) {
            System.err.println("Could not listen on port: " + 4444 + ", " + e.getMessage());
            System.exit(1);
        }

        while (listening) {
            Socket clientSocket = null;
            try {
                clientSocket = serverSocket.accept();
                // System.out.println(clientSocket.getInetAddress());
            } catch (IOException e) {
                System.err.println("Accept failed: " + 4444 + ", " + e.getMessage());
                continue;
            }
        }
    }
}
new KKMultiServerThread(clientSocket).start();

try {
    serverSocket.close();
} catch (IOException e) {
    System.err.println("Could not close server socket."
                     + e.getMessage());
}

B.3.2  KKMultiServThread.java

This is source code for the GUI Server file KKMultiServThread.java

/*@ 
 * Copyright (c)1995, 1996 Sun Microsystems, Inc. All Rights Reserved.
 * 
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies. Please refer to the file "copyright.html"
 * for further important copyright and licensing information.
 * 
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
 * SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT
 * NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE
 * LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING,
 * MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */
import java.net.*;
import java.io.*;

class KKMultiServerThread extends Thread {
    Socket socket = null;

    KKMultiServerThread(Socket socket) {
super("KKMultiServerThread");
this.socket = socket;
}

public void run() {
    try {
        DataInputStream is = new DataInputStream(           
           new BufferedInputStream(socket.getInputStream()));

        KKState kks = new KKState();
        String inputLine;
        String[] outputLine = new String[50];
        int i;

        outputLine = kks.processInput(null, socket);
        // for (i=0; i < outputLine.length; ++i ){
        //     for (i=0; i < 50; i++){
        //         os.println(outputLine[i]+i);
        //     }
        //     os.flush();
        // while ((inputLine = is.readLine()) != null) {
        outputLine = kks.processInput(inputLine, socket);
        //     for (i=0; i < outputLine.length; ++i ){
        //         os.println(outputLine[i] +"My butt smells");
        //     }
        //     os.flush();
        if (outputLine.equals("Bye"))
            break;
        //    os.close();
        //    is.close();
        //    socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

B.3.3 KKState.java

This is source code for the GUI Server file KKState.java

	/*
 * Copyright (c) 1995, 1996 Sun Microsystems, Inc. All Rights
import java.net.*;
import java.io.*;
import java.lang.*;
import java.awt.*;

class KKState {
    Socket socket = null;

    // KKState(Socket socket){
    //      super("KKState");
    //      this.socket = socket;
    //    }

    String[] processInput(String theInput, Socket socket) {
        String[] briReturn = new String[50];
        String theOutput = null;

        theOutput = theInput;
        //        System.out.println("Exec got "+theOutput);
        briReturn = Exec.execPrint("./fe "+theInput, socket);
        return briReturn;
    }
}
import java.net.*;
import java.io.*;
/** A class that eases the pain of running external processes from
 * applications.
 * Lets you run a program three ways:
 * <UL>
 * <LI><B>exec</B>: Execute the command, returning immediately
 * even if the command is still running. This would be
 * appropriate
 * for printing a file.
 * <LI><B>execWait</B>: Execute the command, but don't return
 * until the command finishes. This would be appropriate for
 * sequential commands where the first depends on the second
 * having finished (e.g. <CODE>javac</CODE> followed by
 * <CODE>java</CODE>.
 * <LI><B>execPrint</B>: Execute the command and print the output.
 * This would be appropriate for the UNIX command <CODE>ls</CODE>.
 * </UL>
 * Note that, at least as of JDK 1.0, the path is not taken into
 * account, and you must specify the <B>full</B> pathname to
 * the command, and shell builtin commands will not work.
 * For instance, on UNIX the above three examples might look like:
 * <UL>
 * <LI><PRE>Exec.exec("/usr/ucb/lpr Some-File");</PRE>
 * <LI><PRE>Exec.execWait("/usr/local/bin/javac Foo.java");
 * Exec.execWait("/usr/local/bin/java Foo");</PRE>
 * <LI><PRE>Exec.execPrint("/usr/bin/ls -al");</PRE>
 * </UL>
 * The latest version of the source is at
 * version of the documentation is at
 * No warranty of any kind is provided (obviously :-).
public class Exec {
    //-------------------
    public static Socket socket = null;
    //----------Added by Bri----------------------
    private static boolean verbose = true;

    /** Determines if the Exec class should print which
     * commands are being executed, and print error messages if
     * a problem is found. Default is true.
     * @param verboseFlag true: print messages. False: don't.
     */
    public static void setVerbose(boolean verboseFlag) {
        verbose = verboseFlag;
    }

    //----------------------------------------------
    /** Starts a process to execute the command. Returns immediately,
     * even if the new process is still running.
     * @param command The <B>full</B> pathname of the command to be
     * executed. No shell builtins or shell meta-chars allowed.
     * @return false if a problem is known to occur, but since this
     * returns immediately, problems aren't usually found.
     * Returns true otherwise.
     */
    public static String[] exec(String command) {
        return(exec(command, false, false));
    }

    //---------------------------------------------------
/** Starts a process to execute the command.
 * Waits for the process to
 * finish before returning.
 *
 * @param command The <B>full</B> pathname of the command to be
 * executed. No shell builtins or shell meta-chars allowed.
 * @return false if a problem is known to occur, either due to
 * an exception or from the subprocess returning a non-zero value.
 * Returns true otherwise.
 */

public static String[] execWait(String command) {
    return(exec(command, false, true));
}

//**Starts a process to execute the command. Prints all output the
 * command gives.
 * 
 * @param command The <B>full</B> pathname of the command to be
 * executed. No shell builtins or shell meta-chars allowed.
 * @return false if a problem is known to occur, either due to
 * an exception or from the subprocess returning a non-zero value.
 * Returns true otherwise.
 */

public static String[] execPrint(String command, Socket client_socket) {
    socket = client_socket;
    return(exec(command, true, false));
}

//-------------------------------

private static String[] exec(String command, boolean printResults, boolean wait) {

    String[] briReturn = new String[50];
    int i = 0;

    if (verbose) {
        printSeparator();
        //System.out.println("Executing '" + command + '".");
        //System.out.println(socket.getInetAddress());
    }
}
try {
  PrintStream os = new PrintStream(new BufferedOutputStream(socket.getOutputStream(), 1024), false);
  Process p = Runtime.getRuntime().exec(command);
  if(printResults) {
    DataInputStream commandResult =
      new DataInputStream(new BufferedInputStream(p.getInputStream()));
    String s = null;
    try {
      while ((s = commandResult.readLine()) != null) {
        //briReturn[i] = s;
        i++;
        os.println(s);
        os.flush();
        //System.out.println("Output: " + s);
      }
      if (p.exitValue() != 0) {
        if (verbose)
          printError(command + " -- p.exitValue() != 0");
        briReturn[0]="false";
        return(briReturn);
      }
    } catch (Exception e) {}  
  } else if (wait) {
    try {
      // Doesn’t always wait. If the previous exec
      // was a print-the-results
      // version, then this will NOT wait unless there is a
      // System.out.println call here! Odd...
      System.out.println(" ");
      int returnVal = p.waitFor();
      if (returnVal != 0) {
        if (verbose)
          printError(command);
        briReturn[0]="false";
        return(briReturn);
      }
    } catch (Exception e) {
      if (verbose)
        printError(command, e);
      briReturn[0]="false";
      return(briReturn);
catch (Exception e) {
    if (verbose)
        printError(command, e);
    briReturn[0] = false;
    return (briReturn);
}

//System.out.println(briReturn);
return (briReturn);

private static void printError(String command, Exception e) {
    System.out.println("Error doing exec(" + command + "): " + e.getMessage());
    System.out.println("Did you specify the full pathname?");
}

private static void printError(String command) {
    if (!command.equals("./fe null -- p.exitValue() != 0"))
        System.out.println("Error executing '" + command + ")
}

private static void printSeparator() {
//    System.out.println
//        ("==================================================================");
}
B.4 PERL Resender Source Code

This is the source code for the Resender program, in PERL

#!/usr/local/perl/perl -U

# Copyright 1992, D. Brent Chapman. See the Majordomo
# license agreement for usage rights.
# Modified By Brian Wilson to read the Mailing list files
# From the right directories
#
#$Source: /sources/cvsrepos/majordomo/resend,v$
#$Revision: 1.28.2.2.2.3$
#$Date: 1995/01/07 17:31:43$
#$Author: rouilj$
#$State: Exp$
#
#$Locker: $
#
# updated to
# Revision: 1.19
# Date: 1993/11/11 02:23:37
#
#
# set our path explicitly
$ENV{'PATH'} = "/bin:/usr/bin:/usr/ucb";

# What shall we use for temporary files?
$tmp = "/tmp/majordomo.$$";

# Before doing anything else tell the world I am resend
# The mj_prefix is reserved for tools that are part
# of majordomo proper.
$main'program_name = 'mj_resend';

# If the first argument is "@filename", read the real arguments
# from"filename", and shove them onto the ARGV for later processing
# by &Getopts()

if ($ARGV[0] =~ /^@/) {
    $fn = shift($ARGV);
    $fn =~ s/^@//;
    open(AV, $fn) || die("open(AV, "$fn") : "$!\nStopped");
undef($/);  # set input field separator
$av = <AV>;  # read whole file into string
close(AV);
@av = split(/\s+/, $av);
unshift(@ARGV, @av);
$/ = "\n";
}

# Read and execute the .cf file
$cf = $ENV{"MAJORDOMO_CF"} || "/etc/majordomo.cf";
if ($ARGV[0] eq "-C") {
    $cf = $ARGV[1];
    shift(@ARGV);
    shift(@ARGV);
}
if (! -r $cf) {
    die("$cf not readable; stopped");
}
eval('cat $cf') || die 'eval of majordomo.cf failed';

chdir($homedir) || die("Can't chdir(""$homedir"": $!");
unshift(@INC, $homedir);
require "majordomo.pl";
require "majordomo_version.pl";
require "getopts.pl";
require "config_parse.pl";

    || die("resend: Getopts() failed: $!");

if (! defined($opt_l) || ! defined($opt_h)) {
    die("resend: must specify both '-l list' and
         '-h host' arguments");
}

# smash case for the list name
$opt_l = ~ tr/A-Z/a-z/;
if ( ! @ARGV) {
    die("resend: must specify outgoing list as last arg(s)");
}

$opt_r = "$opt_r@$opt_h" if ( defined($opt_r) );

&get_config($listdir, $opt_l);
$opt_A = &cf_ck_bool($opt_l,"moderate")
    if &cf_ck_bool($opt_l,"moderate")
$opt_h = $config_opts($opt_l,"resend_host")
    if($config_opts($opt_l,"resend_host") ne '');
print $config_opts($opt_l, "approve_passwd");

$opt_a = $config_opts($opt_l,"approve_passwd")
    if ($config_opts($opt_l,"approve_passwd") ne ' ');
$opt_M = $config_opts($opt_l,"maxlength")
    if ($config_opts($opt_l,"maxlength") ne ' ');
$opt_f = $config_opts($opt_l,"sender")
    if ($config_opts($opt_l,"sender") ne ' ');
$opt_p = $config_opts($opt_l,"precedence")
    if ($config_opts($opt_l,"precedence") ne ' ');
$opt_r = $config_opts($opt_l,"reply_to")
    if ($config_opts($opt_l,"reply_to") ne ' ');
$opt_I = $config_opts($opt_l,"restrict_post")
    if ($config_opts($opt_l,"restrict_post") ne ' ');
$opt_R = &cf_ck_bool($opt_l,"purge_received")
    if &cf_ck_bool($opt_l,"purge_received");
$opt_s = &cf_ck_bool($opt_l,"administrivia")
    if &cf_ck_bool($opt_l,"administrivia");
$opt_d = &cf_ck_bool($opt_l,"debug")
    if &cf_ck_bool($opt_l,"debug");

if (defined($opt_f)) {
    $sendmail_sender = $opt_f;
} else {
    $sendmail_sender = "$opt_l-request";
}

if (defined($opt_a)) {
    if ($opt_a =~ /~\//) {
        open(PWD, $opt_a) || die("resend: open(PWD, "$opt_a"): $!");
        $opt_a = &chop_nl(<PWD>);
    }
# else {
#   $thefile = "/home/bwilson/src/mailer/mailer/
#             tmp/listnames/passwd/".$opt_a;
#   open(PWD, $thefile) || die("resend: 
#       open(PWD, "$thefile"): $!");
#   $opt_a = &chop_nl(<PWD>);
# }

if (defined($opt_A) &amp; ! defined($opt_a)) {
  die("resend: must also specify '-a passwd' if using '-A' flag");
}

$sender = "$sendmail_sender@$opt_h";

&open_temp(OUT, "/tmp/resend.$$.out") ||
  &abort("resend: Can't open /tmp/resend.$$.out: $!");

&open_temp(IN, "/tmp/resend.$$.in") ||
  &abort("resend: Can't open /tmp/resend.$$.in: $!");

while (<STDIN>) {
  print IN $-;
}

close(IN);

open(IN, "/tmp/resend.$$.in") ||
  die("resend: Can't open /tmp/resend.$$.tmp: $!");

do {
  $restart = 0;
  $pre_hdr = 1;
  while (<IN>) {
    if ($pre_hdr) {
      if (/\s*$/) {
        # skip leading blank lines; usually only there if this is a
        # restart after an in-body "Approved:" line
        next;
      } else {
        $pre_hdr = 0;
        $in_hdr = 1;
        $kept_last = 0;
      }
    }
    if ($in_hdr) {
      #
if (/*s*/)
{
    # end of header; add new header fields
    print OUT "Sender: $sender\n";
    if (defined($opt_p)) {
        print OUT "Precedence: $opt_p\n";
    }
    if (defined($opt_r)) {
        print OUT "Reply-To: ",
        &config'substitute_values($opt_r),
        "\n";
    }

    # print out additional headers
    if ( $config_opt="message_headers" ne ' ' ) {
        local($headers) = &config'substitute_values ( $config_opt="message_headers", $opt_l);
        $headers =~ s/\001\n/g;
        print OUT $headers;
    }

    $in_hdr = 0;
    print OUT "$_.

    # print out front matter
    if ( $config_opt="message_fronter" ne ' ' ) {
        local($fronter) = &config'substitute_values ( $config_opt="message_fronter", $opt_l);
        $fronter =~ s/\001\$/\n/g
        print OUT $fronter;
    }
} elsif (/*from */ && defined($opt_a)) {
    $approved = &chop_n1($1);
    if ($approved ne $opt_a &&
        !(&main'valid_passwd($listdir, $opt_l, $approved))) {
        &bounce("Invalid 'Approved:' header");
    }
} elsif (/*from */
          # skip all these headers
          
          sender:/
          return-receipt-to:/
          errors-to:/
          return-path:/
          (/*reply-to:/ && defined($opt_r))
          # skip only if "-r" set
          (/*precedence:/ && defined($opt_p))
          # skip only if "-p" set
|| (\^-received:/i && defined($opt_R))
  # skip only if "-R" set
|| (\^-s/ && ! $kept_last)
  # skip if skipped last
} {
  # reset $kept_last in case next line is continuation
  $kept_last = 0;
} else {
  # check for administrivia requests
  if (defined($opt_s) && ! defined($approved)
      && (\^-subject:\s*subscribe\b/i ||
          /^subject:\s*unsubscribe\b/i ||
          /^subject:\s*help\b/i ||
          /^subject:\s*RCPT:\b/ ||
          /^subject:\s*Delivery Confirmation\b/ ||
          /^subject:\s*NON-DELIVERY of:/ ||
          /^subject:\s*Undeliverable Message\b/ ||
          /^subject:\s*Receipt Confirmation\b/ ||
          /^subject:\s*Failed mail\b/ ||
          /^subject:\s.*\bchange\b.*\baddress\b/ ||
          /^subject:\s*request\b.*\baddition\b/i))
      &bounce("Admin request");
}

# prepend subject prefix
if ( (\^-subject:\s*/i) &&
     ($config_opts\$opt_1,"subject_prefix") ne '')
  ) {
  local($foo) = &config'substitute_values(\n    $config_opts\$opt_1,"subject_prefix"), $opt_1);\n  local($foo_pat) = $foo;
  $foo_pat = s/(\W)/\$1/g;
  s/^subject:\s*/Subject: $foo /i if !/$foo_pat/;
}

if ( /^from:\s*(.+)/i )
  {
    $from = $1;
    $from_last = 1;
  }
elsif ( defined($from_last) )
  {
    if ( /^s+(.+)/ )
      {
        $from = " $1";
      }
else
{
    undef($from_last);
}

&check_hdr_line($_);  # check for length & balance
$kept_last = 1;
print OUT $_;
}

# this isn't a header line, so print it (maybe)
# first, though, is the first line of the body
# an "Approved:" line?
if (($body_len == 0) && /^approved:\s*\([^\s].\)/i
    && defined($opt_a)) {
    # OK, is it a valid "Approved:" line?
    $approved = &chop_nl($1);
    if ($approved ne $opt_a &&
        !(&main'valid_passwd($listdir, $opt_l,
            $approved))) {
        &bounce("Invalid 'Approved:' header");
    } else {
        # Yes, it's a valid "Approved:" line...
        # So, we start over
        $restart = 1;
        close(OUT);
        unlink("/tmp/resend.$$.out");
        &open_temp(OUT, "/tmp/resend.$$.out") ||
        &abort("resend:2 Can't open /tmp/resend.
            $$ .out: $!");

        last;
    }
}

$body_len += length($_);
# make sure it doesn't make the message too long
if (defined($opt_M) && ! defined($approved)
    && ($body_len > $opt_M)) {
    &bounce("Message too long (> $opt_M"));
}

# add admin-request recognition heuristics here. (body)
if (defined($opt_s) && ! defined($approved)
    && ($body_line++ < 5) &&
    /^(bad\s+me)\b/i
    || /^(bdelete me)\b/i || /^bremove\s+me\b/i
    || /^bchange\b.*\baddress\b/
    || /^bsubscribe\b/i || /^sub\b/i
while ($restart);

if ( $config_opts{opt_1,"message_footer"} ne '' ) {
    local($footer) = &config'substitute_values($config_opts{opt_1,"message_footer"}, $opt_1);
    $footer =~ s/\001\n/g;
    print OUT $footer;
}

close(OUT);

if ( defined($opt_1) && defined($from) && ! defined($approved) ) {
    local($infile) = 0;

    @files = split ( "[\t\n]+", $opt_1);

    foreach $file (@files) {
        if ($file !~ /\//) {
            $file = "$listdir/$file";
        }
    }
if (open (LISTFD, "<$file") != 0) {
    @output = grep (&addr_match($from, $_), <LISTFD>);
    close (LISTFD);

    if ($#output != -1) {
        $infile = 1;
        last;
    }
} else {
    die("resend:a Can’t open $file: $!");
}

if ($infile == 0) {
    &bounce ("Non-member submission from [$from]");
}

if (defined($opt_A) && ! defined($approved)) {
    &bounce ("Approval required");
}

/sendmail_cmd = "/usr/lib/sendmail $opt_m -f/sendmail_sender " .
    join (" ", @ARGV);

if (defined($opt_d)) {
    $! = 1;
    print "Command: $sendmail_cmd\n";
    $status = (system("cat /tmp/resend.$$ .out") >> 8);
    unlink(</tmp/resend.$$ .>*);
    exit($status);
} else {
    local (*MAILOUT, *MAILIN, @mailer);
    @mailer = split (" ", "$sendmail_cmd");
    open (MAILOUT, "|-") || &do_exec_sendmail(@mailer);
    open (MAILIN, "/tmp/resend.$$ .out");
    while (<MAILIN>) {
        print MAILOUT $_
    }
    close(MAILOUT);
    close(MAILIN);
    unlink(</tmp/resend.$$ .>*);
    exit(0);
}

sub check_balance {
# set a temporary variable
local($t) = shift;

# strip out all nested parentheses
1 while $t =~ s/\([\^\(]*\)/\1/g;

# strip out all nested angle brackets
1 while $t =~ s/\<[^\<]*\>/\1/g;

# if any parentheses or angle brackets remain, were imbalanced
if ($t =~ /\(\)/\<\>/ && ! defined($approved)) {
    &bounce("Imbalanced parentheses or angle brackets");
    return(undef);
}
return(1);

sub check_hdr_line {
    local($_) = shift;

    if (! /\s/) {  # is this a continuation line?
        # Not a continuation line.
        # If $balanced_fld is defined, it means the last field was one
        # that needed to have balanced "()" and "<>" (i.e., "To:", "From:",
        # and "Cc:", so check it. We do it here in case the last field was
        # multi-line.

        if (defined($balanced_fld)) {
            &check_balance($balanced_fld);
        }
    }
    # we undefine $balanced_fld and reset $field_len; these may be set
    # below

    undef($balanced_fld);
    $field_len = 0;
}

    # is this a field that must be checked for balanced "()" and "<>"?
if (defined($balanced_fld) || /^from:/i || /^cc:/i || /^to:/i) {
    # yes it is, but we can't check it yet because there might be
    # continuation lines. Buffer it to be checked at the beginning
    # of the next non-continuation line.
    # is this line too long?
if ((length($_) > 128) && ! defined($approved)) {
    &bounce("Header line too long (>128)");
    return(undef);
}
# is this field too long?
if ((($field_len += length($_.)) > 1024)
    && ! defined($approved)) {
    &bounce("Header field too long (>1024)");
    return(undef);
}

$balanced fld = $_.;
chop($balanced fld);
}

# if we get here, everything was OK.
return(1);
}

sub bounce {
    local($reason) = shift;
    local($_.);

    &resend_sendmail(BOUNCE, $sender, "BOUNCE $opt_l@$opt_h:
                        $reason");

    seek(IN, 0, 0);
    while (<IN>) {
        print BOUNCE $_.;
    }
    close(BOUNCE);
    unlink(</tmp/resend.$$.*>);
    exit(0);
}

sub resend_sendmail {
    local(*MAIL) = shift;
    local($to) = shift;
    local($subject) = shift;

    # clean up the addresses, for use on the sendmail command line
    local(@to) = &ParseAddrs($to);
    for (@to) {
        $_ = join("", ", ", &ParseAddrs($_.));
    }
    $to = join("", ", @to);

    # open the process
    if (defined($opt_d)) {
        # debugging, so just say it, don't do it
open(MAIL, ">-");  
    print MAIL ">>> /usr/lib/sendmail -f$sendmail_sender -t\n";
} else {
local(@mailer) = split(' ',"/usr/lib/sendmail
    -f$sendmail_sender -t");
    open(MAIL, "|-") || &do_exec_sendmail(@mailer);
}

    # generate the header
    print MAIL <<"EOM"
    To: $to
    From: $sender
    Subject: $subject

    EOM

    return;
}